LECTURE 15

SEGMENT TREE

Big-O Coding

Website: www.bigocoding.com

Giới thiệu tổng quan

Segment Tree (Cây phân đoạn) là một cấu trúc dữ liệu cây nhị phân đầy đủ dùng để giải quyết các bài toán trên dãy số.

Các thao tác trên Segment Tree:

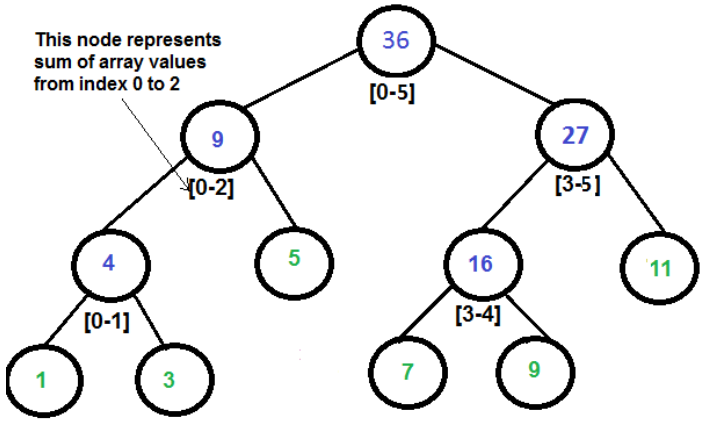
- Query: Truy vấn thông tin một đoạn bất kỳ trên dãy số.
- Update: Cập nhật phần tử bất kỳ trên dãy số.

Một số bài toán ứng dụng:

- Range Minimum Query: Tìm giá trị nhỏ nhất trên một đoạn của dãy số.
- Sum of given range: Tổng một đoạn bất kỳ trên dãy số.

Xây dựng Segment Tree

1. Mỗi node quản lý một đoạn trên mảng.
2. Node cha là **KẾT QUẢ** của 2 node con.
3. Độ phức tạp của phần xây dựng Segment Tree: **O(N)**



Segment Tree for input array {1, 3, 5, 7, 9, 11} GeeksforGeeks

Lưu ý:

- *Nếu bài toán là* **Range Minimum Query** *thì KẾT QUẢ node cha là giá trị nhỏ nhất của 2 node con.*
- *Nếu bài toán là* ***Sum of given range*** *thì KẾT QUẢ node cha là tổng giá trị của 2 node con.*

1. QUERY

RANGE MINIMUM QUERY

Range Minimum Query

Range Minimum Query: Cho mảng gồm N phần tử và Q truy vấn, mỗi truy vấn yêu cầu tìm giá trị nhỏ nhất của một đoạn bất kỳ trên dãy số.

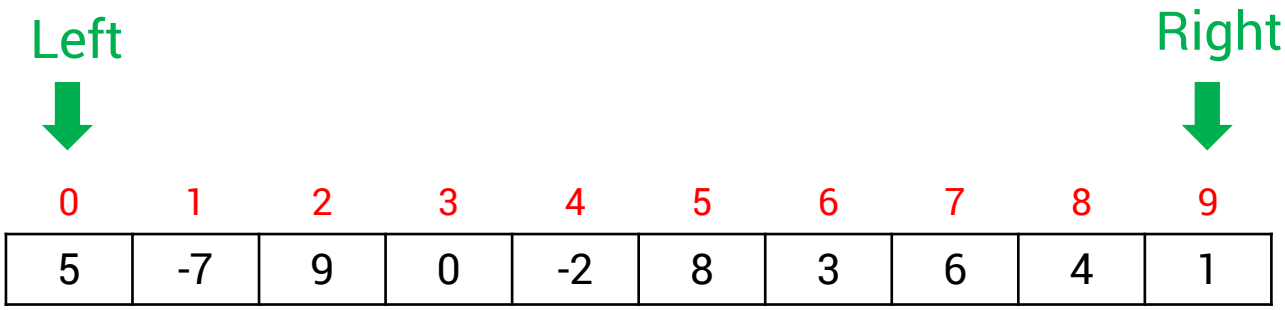
0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1

- Truy vấn [2, 8]: -2
- Truy vấn [0, 4]: -7
- Truy vấn [0, 9]: -7
- ...

Các phương pháp giải quyết:

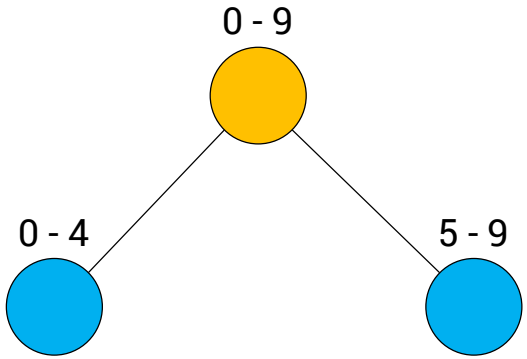
- Dùng Brute Force: độ phức tạp **O(Q*N)**
- Dùng Segment Tree: độ phức tạp **O(N + Q*logN)**

Bước 0: Xây dựng Segment Tree (1)

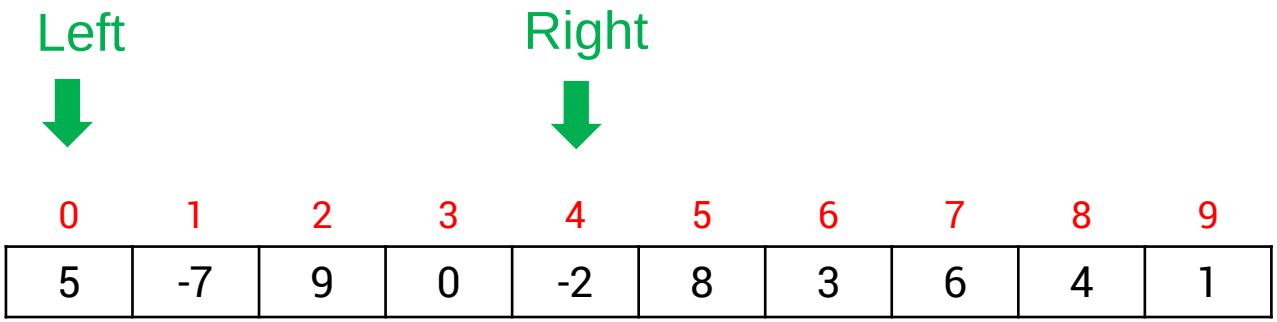


(Node gốc) Phân rã cây nhị phân để thêm giá trị vào:

- Left = 0
- Right = 9
- Mid = (Left + Right)/2 = 4
- Part 1 (node con trái quản lý nửa đoạn bên trái): Left → Mid (**Node 0 – 4**)
- Part 2 (node con phải quản lý nửa đoạn bên phải): Mid + 1 → Right (**Node 5 – 9**)

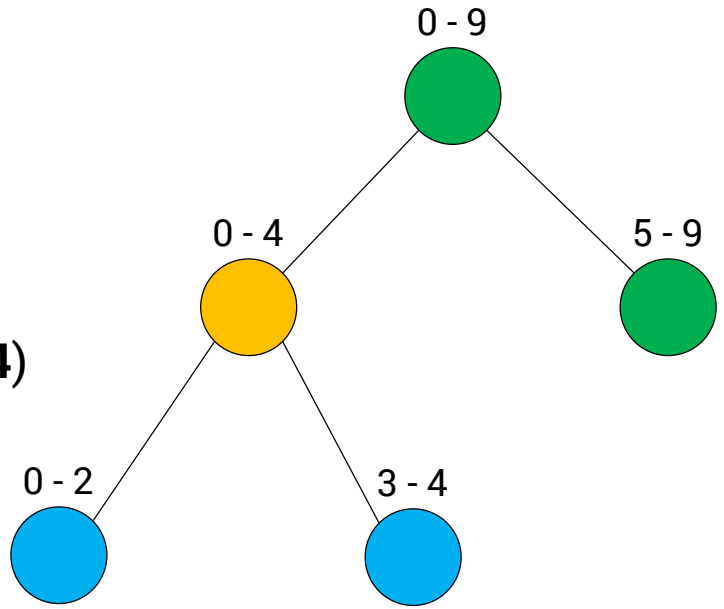


Bước 0: Xây dựng Segment Tree (2)

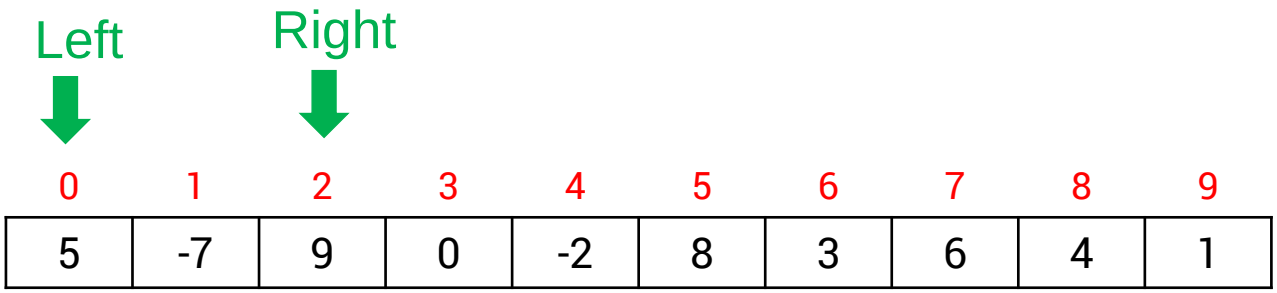


(Node 0 – 4) Phân rã cây nhị phân để thêm giá trị vào:

- Left = 0
- Right = 4
- Mid = (Left + Right)/2 = 2
- Part 1: Left → Mid (**Node 0 – 2**)
- Part 2: Mid + 1 → Right (**Node 3 – 4**)

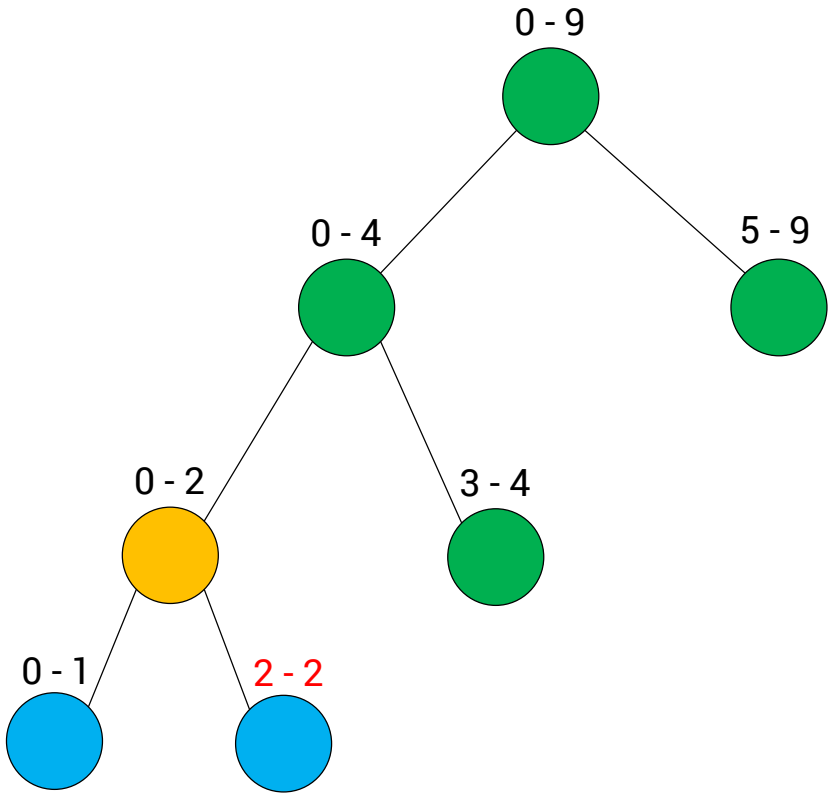


Bước 0: Xây dựng Segment Tree (3)

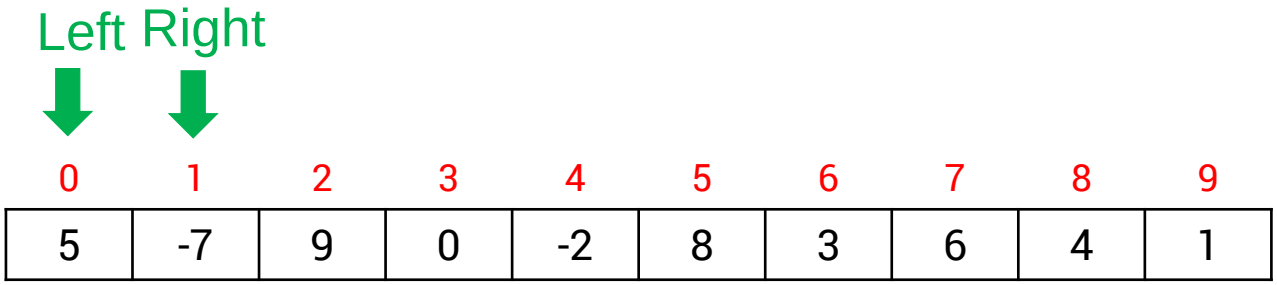


(Node 0 – 2) Phân rã cây nhị phân để thêm giá trị vào:

- Left = 0
- Right = 2
- Mid = (Left + Right)/2 = 1
- Part 1: Left → Mid (**Node 0 – 1**)
- Part 2: Mid + 1 → Right (**Node 2 – 2**)

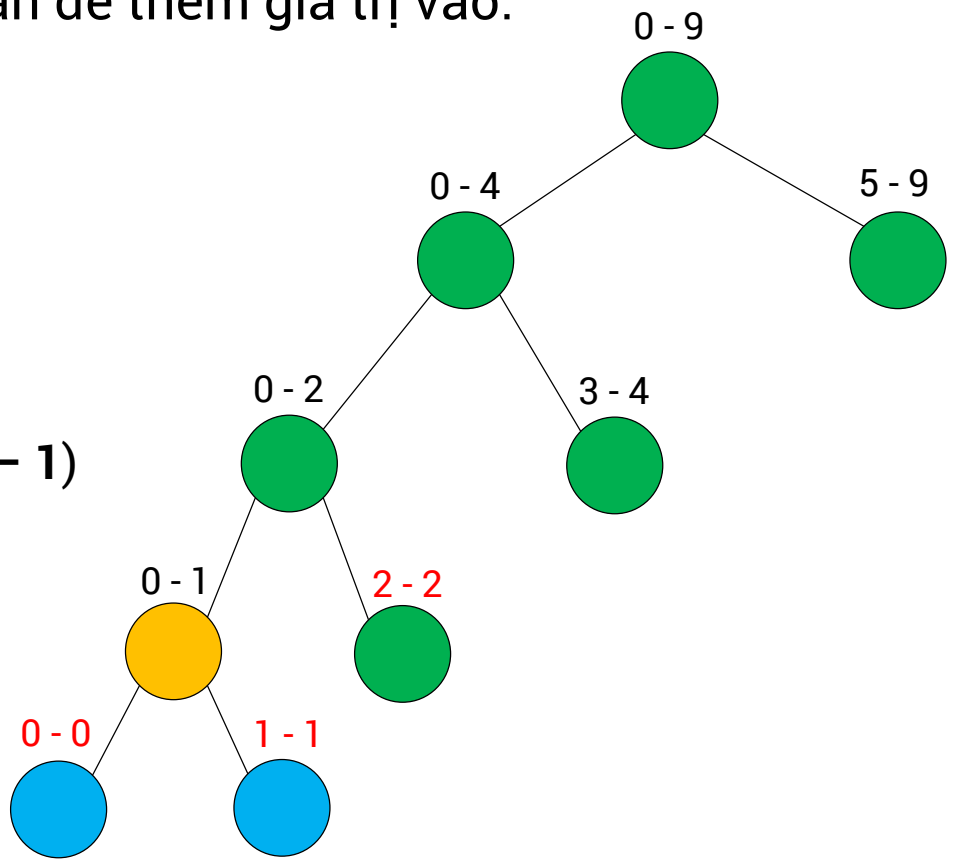


Bước 0: Xây dựng Segment Tree (4)



(Node 0 – 1) Phân rã cây nhị phân để thêm giá trị vào:

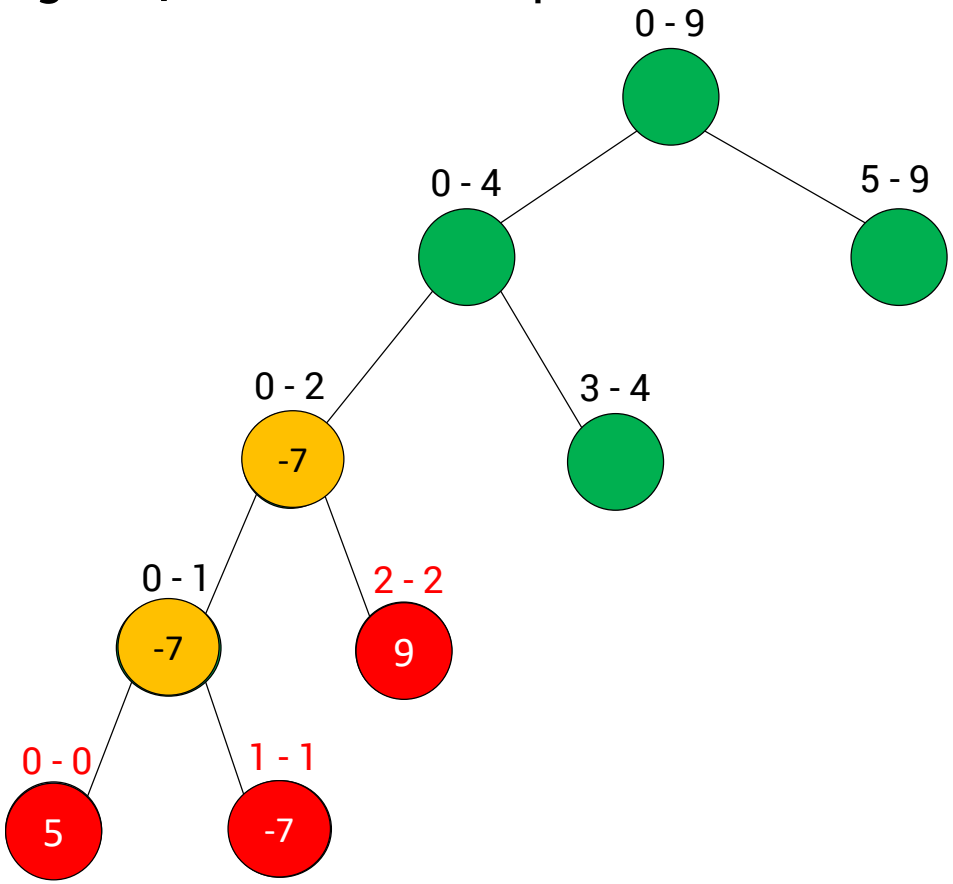
- Left = 0
- Right = 1
- Mid = (Left + Right)/2 = 0
- Part 1: Left → Mid (**Node 0 – 0**)
- Part 2: Mid + 1 → Right (**Node 1 – 1**)



Bước 0: Xây dựng Segment Tree (5)

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1

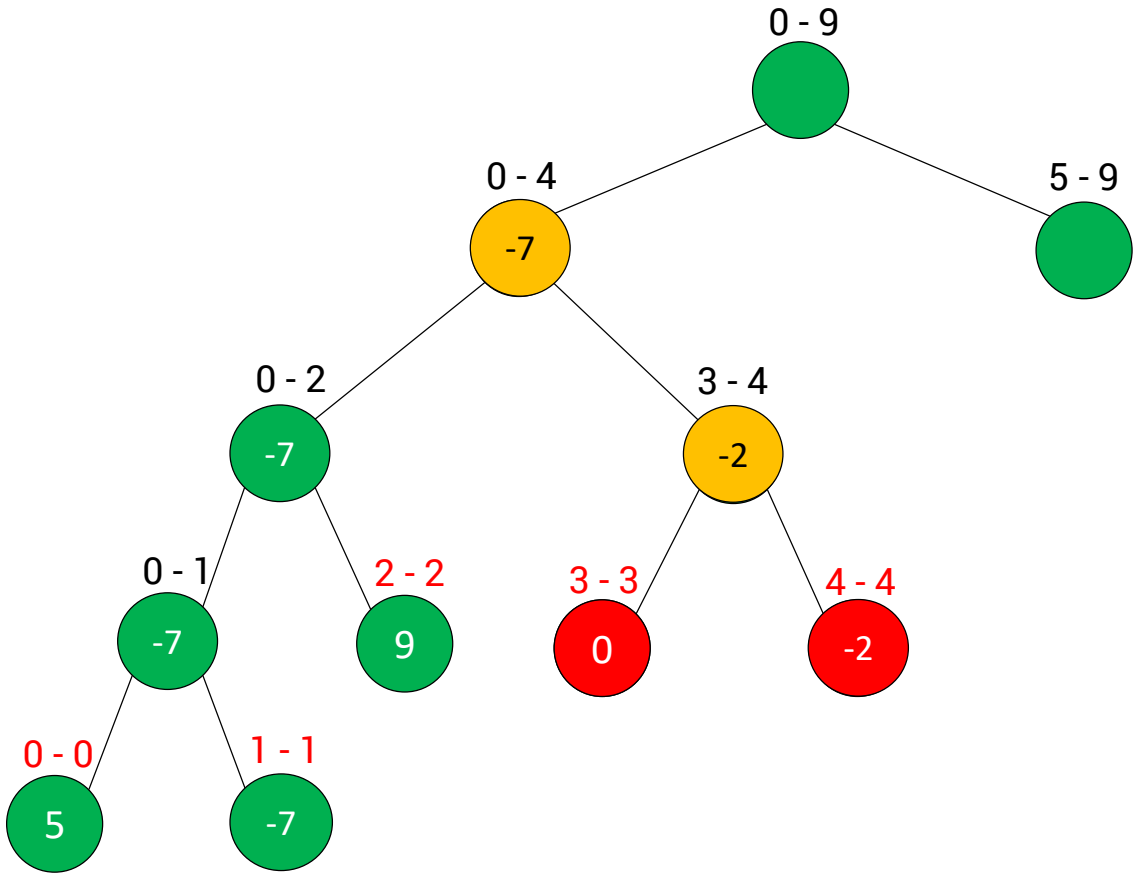
Điền các giá trị ở vị trí 0, 1, 2 vào các node lá của Segment Tree, đồng thời điền giá trị các node liên quan:



Bước 0: Xây dựng Segment Tree (6)

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1

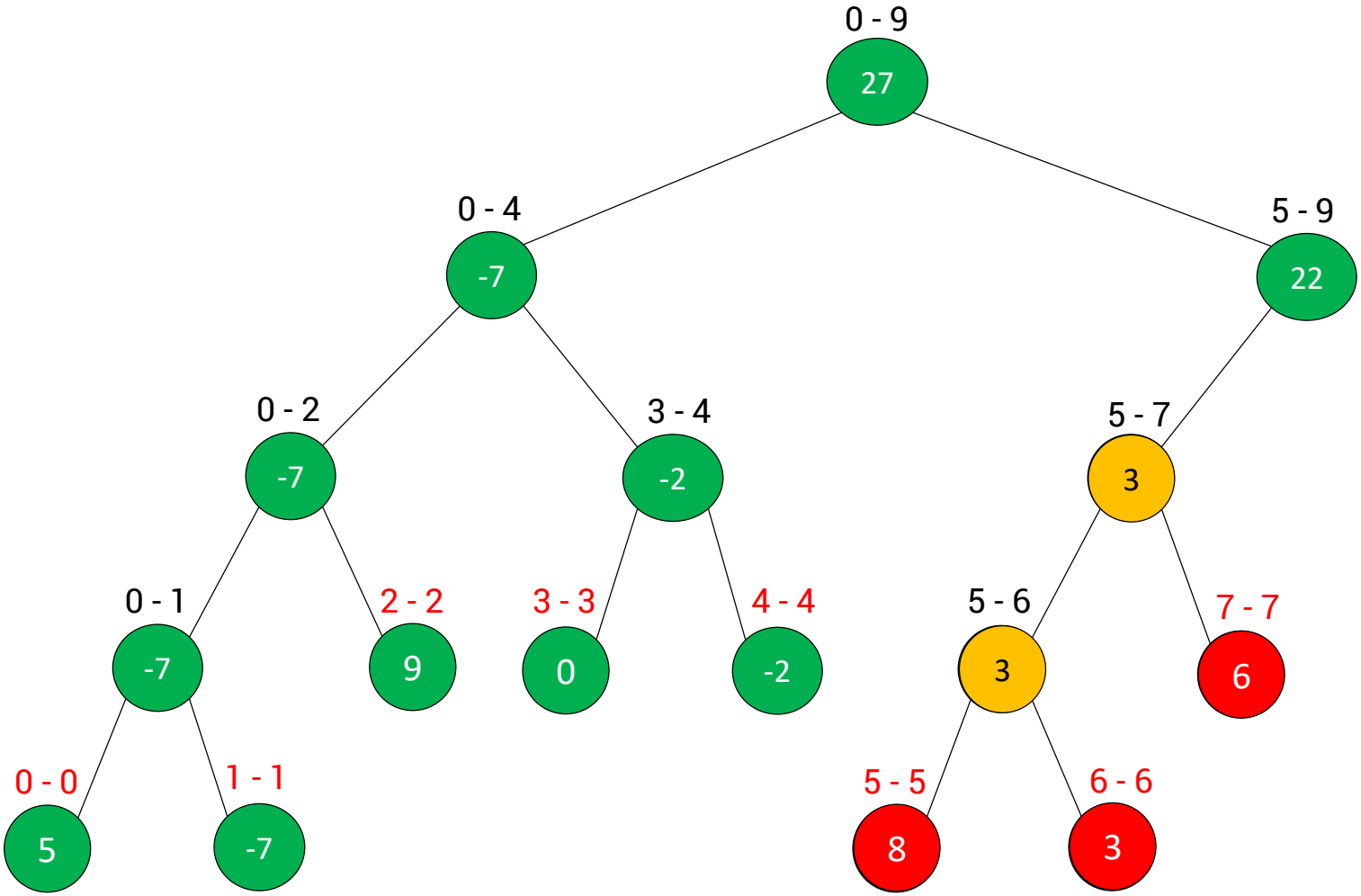
Tương tự phân rã tiếp các node [3, 4] và điền giá trị ở vị trí 3, 4 vào cây, đồng thời điền giá trị các node liên quan:



Bước 0: Xây dựng Segment Tree (7)

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1

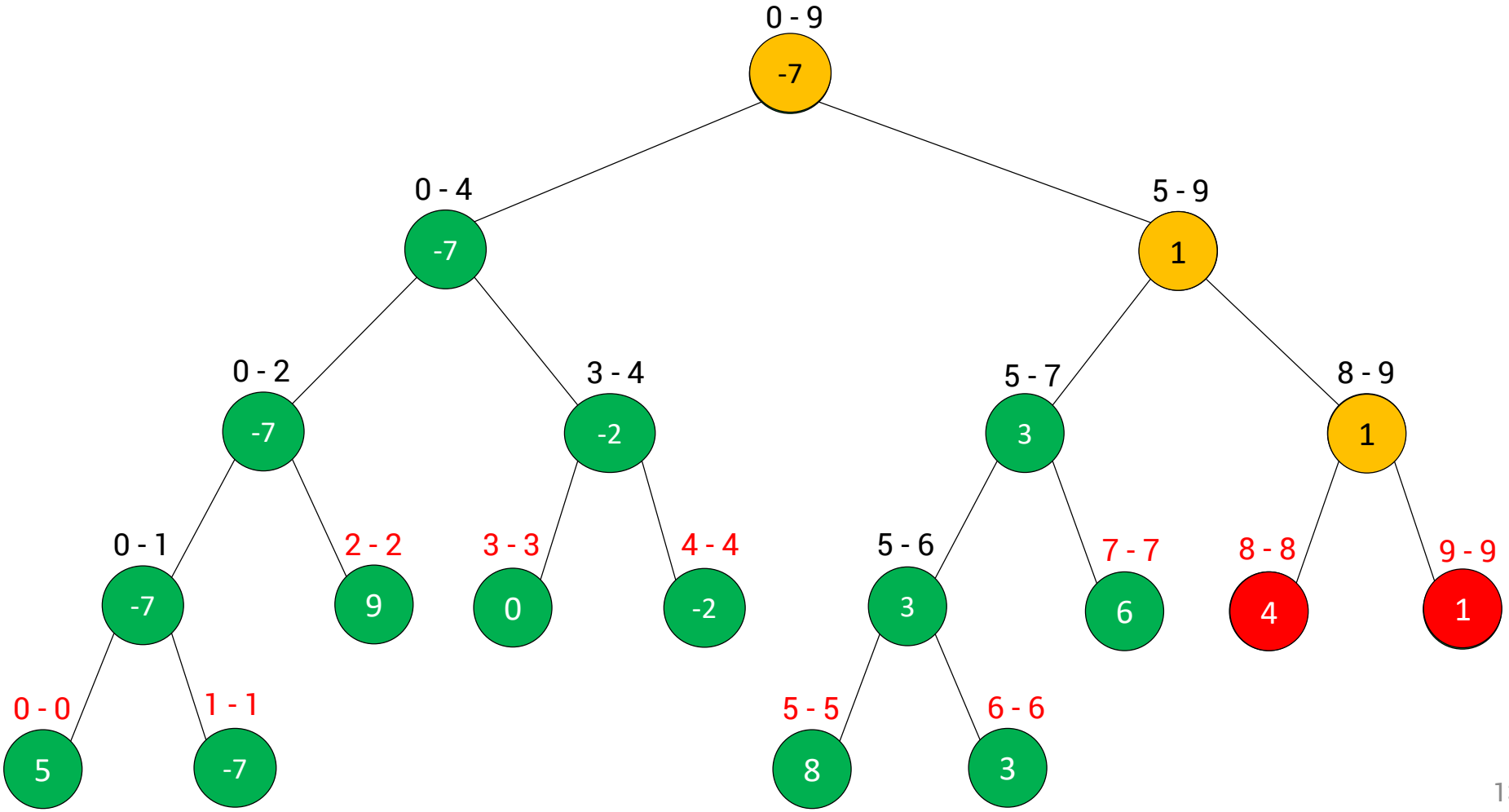
Phân rã và điền các giá trị ở vị trí 5, 6, 7 vào các node lá của cây:



Bước 0: Xây dựng Segment Tree (8)

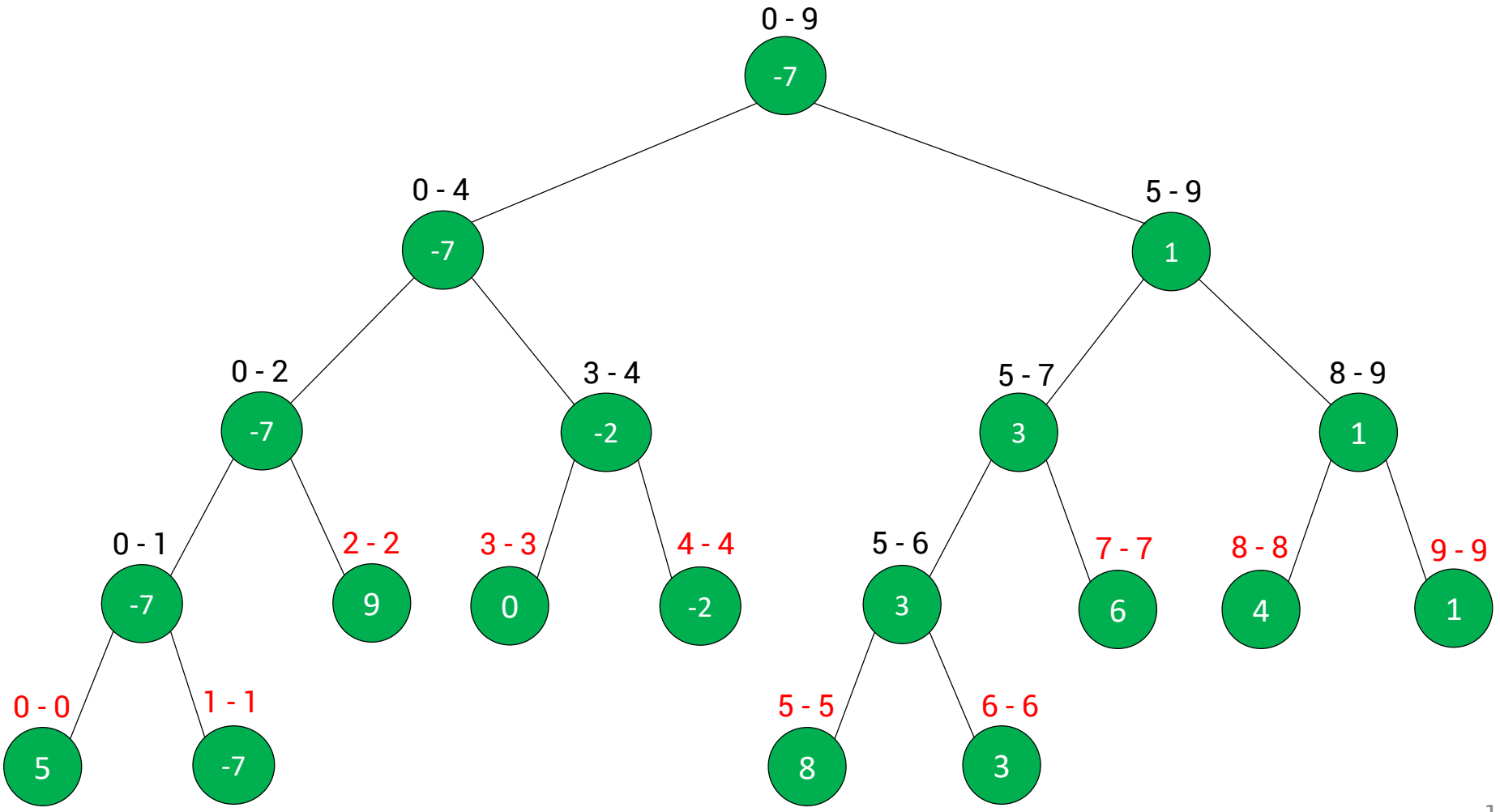
0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1

Phân rã và điền các giá trị ở vị trí 8, 9 vào các node lá của cây:



Bước 0: Kết quả xây dựng cây Segment

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1



Source Code Xây dựng Segment Tree RMQ

```cpp
void buildTree(vector<int> &a, vector<int> &segtree, int left, int right, int index) {
    if (left == right) {
        segtree[index] = a[left];
        return;
    }
    int mid = (left + right) / 2;
    buildTree(a, segtree, left, mid, 2 * index + 1);
    buildTree(a, segtree, mid + 1, right, 2 * index + 2);
    segtree[index] = min(segtree[2 * index + 1], segtree[2 * index + 2]);
}
```

Source Code Xây dựng Segment Tree RMQ

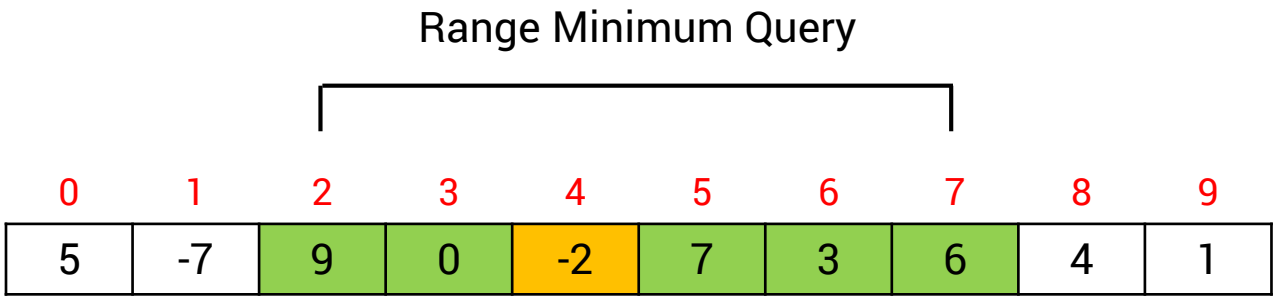
```python
def buildTree(a, segtree, left, right, index):
    if left == right:
        segtree[index] = a[left]
        return
    mid = (left + right) // 2
    buildTree(a, segtree, left, mid, 2 * index + 1)
    buildTree(a, segtree, mid + 1, right, 2 * index + 2)
    segtree[index] = min(segtree[2 * index + 1], segtree[2 * index + 2])
```

Source Code Xây dựng Segment Tree RMQ

```
1.      private static void buildTree(int[] a, int[] segtree, int left, int right,
                                      int index) {
2.          if (left == right) {
3.              segtree[index] = a[left];
4.              return;
5.          }
6.          int mid = (left + right) / 2;
7.          buildTree(a, segtree, left, mid, 2 * index + 1);
8.          buildTree(a, segtree, mid + 1, right, 2 * index + 2);
9.          segtree[index] = Math.min(segtree[2 * index + 1], segtree[2 * index + 2]);
10.     }
```

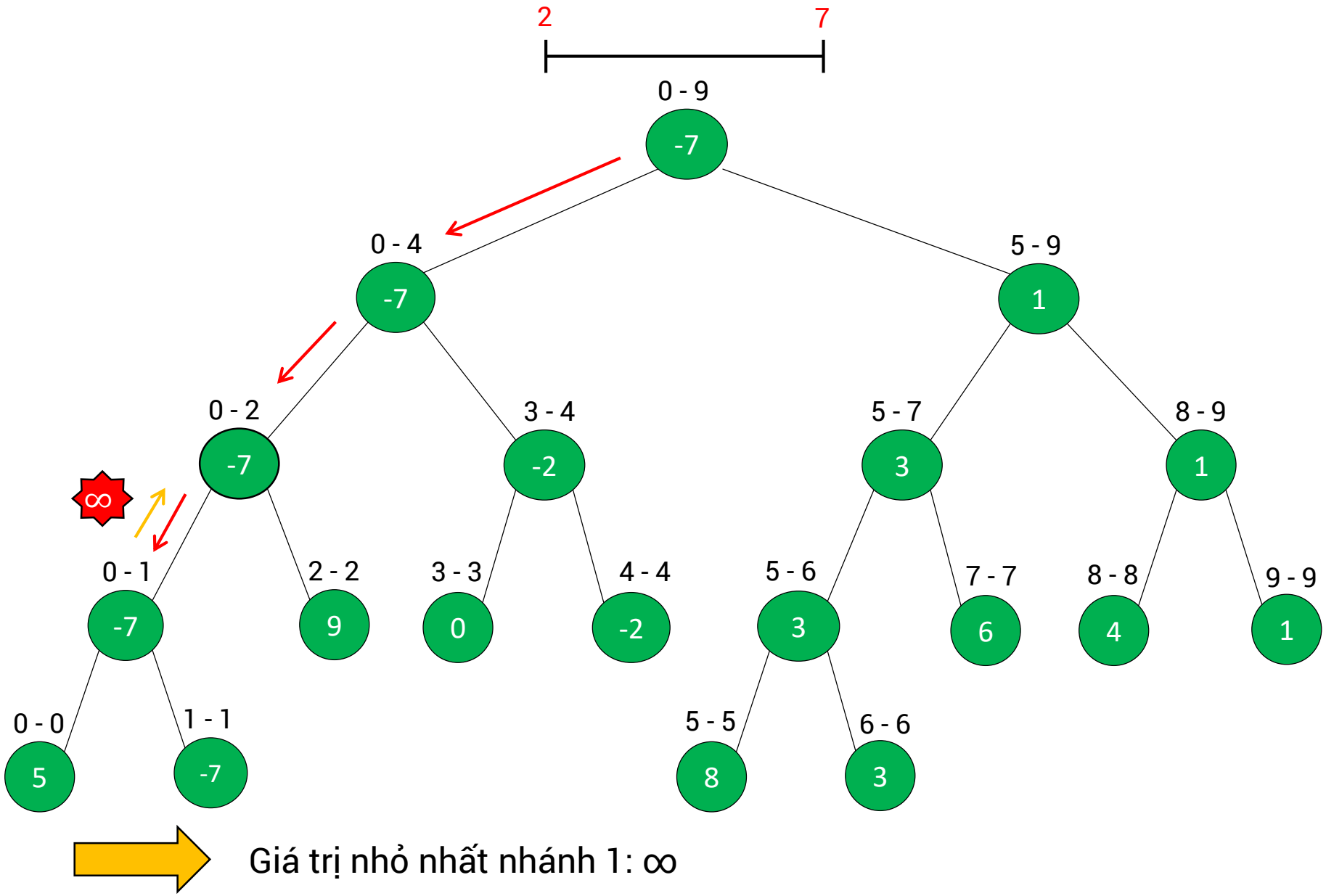
Range Minimum Query

Chạy truy vấn [2, 7] tìm giá trị nhỏ nhất trên đoạn này.



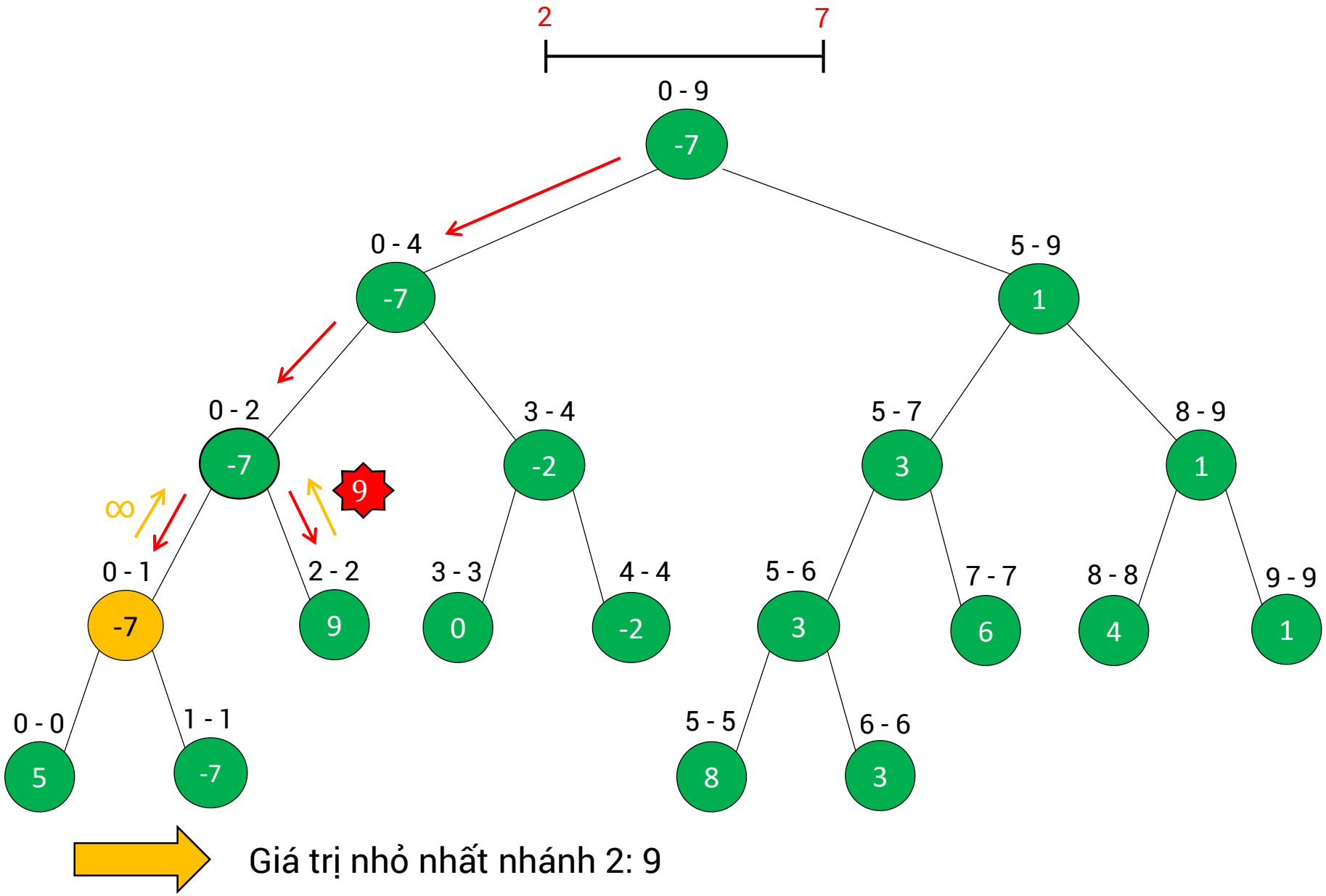
Giá trị nhỏ nhất trong đoạn [2, 7]: -2

Bước 1: Tìm giá trị nhỏ nhất nhánh 1

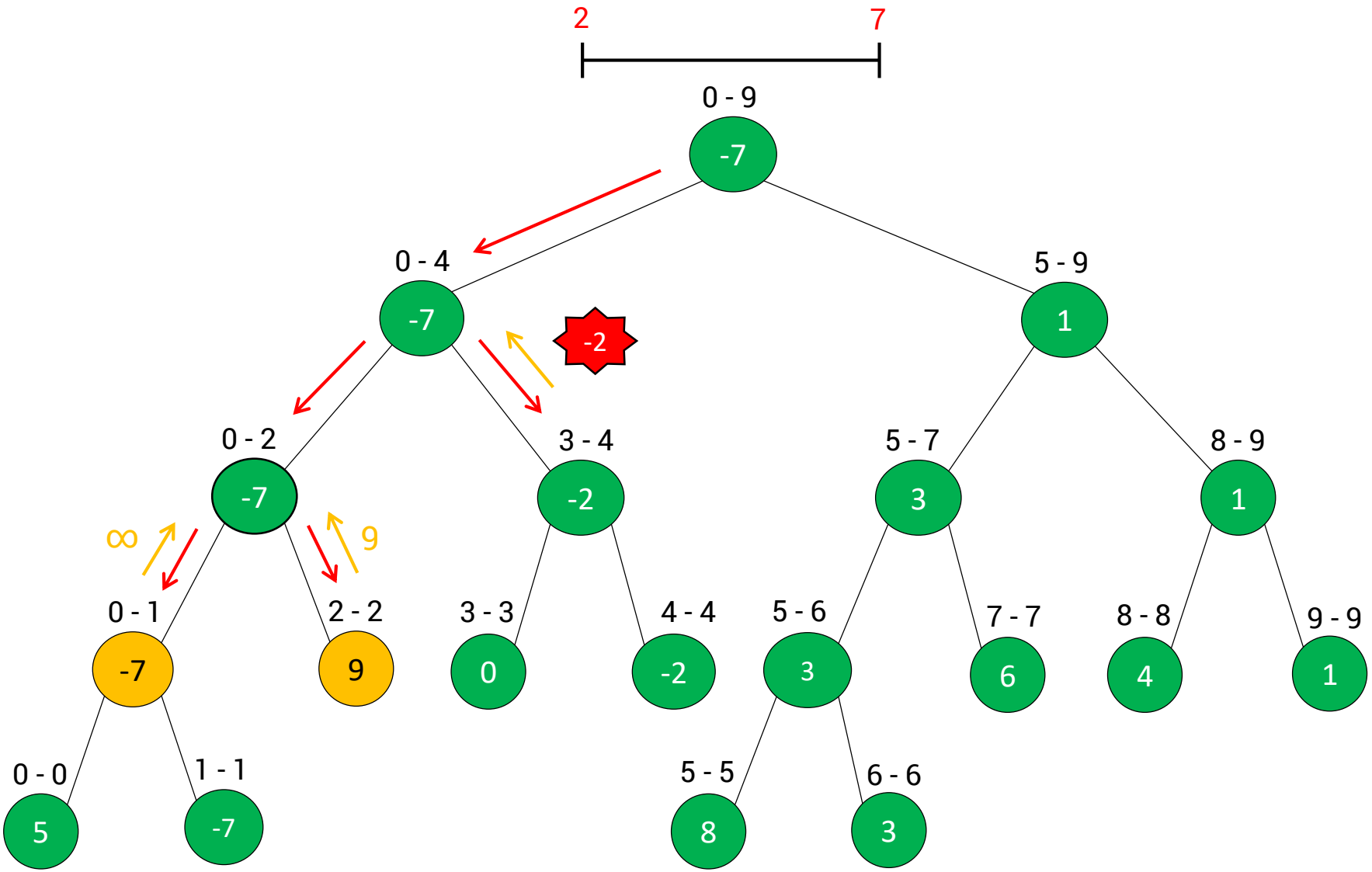


Giá trị nhỏ nhất nhánh 1: ∞

Bước 2: Tìm giá trị nhỏ nhất nhánh 2

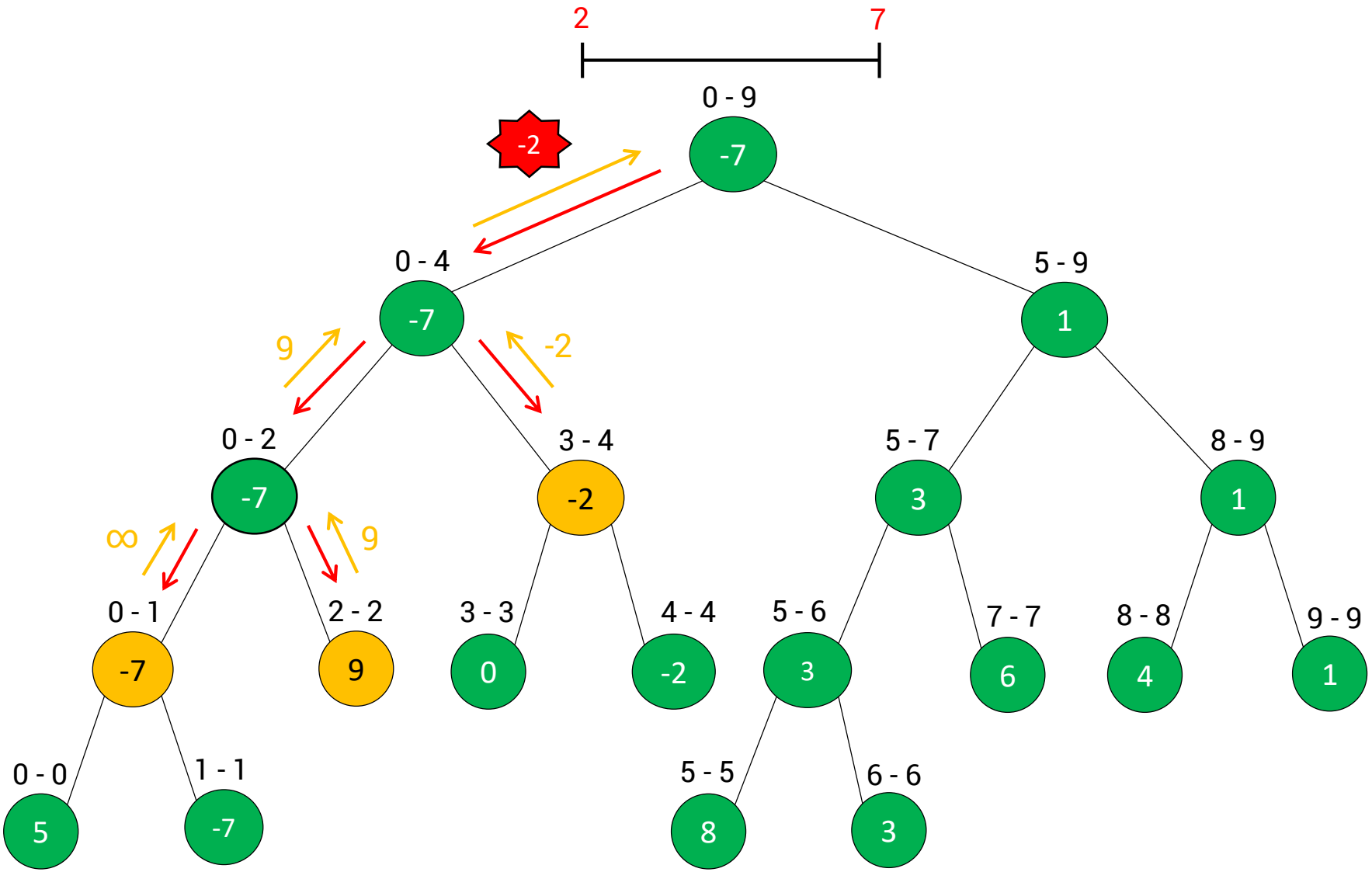


Bước 3: Tìm giá trị nhỏ nhất nhánh 3



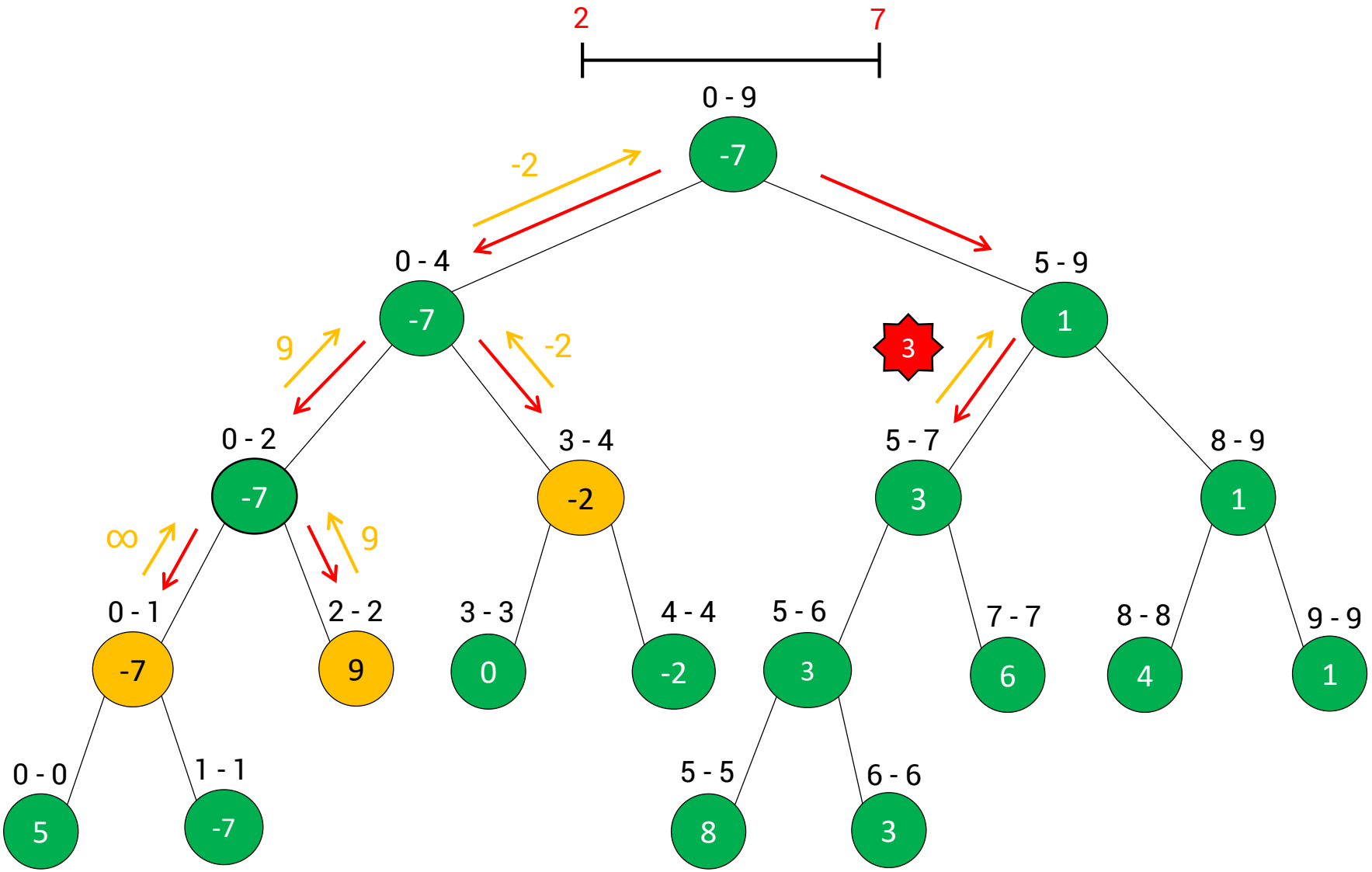
Giá trị nhỏ nhất nhánh 3: -2

Bước 4: Tìm giá trị nhỏ nhất nhánh trái



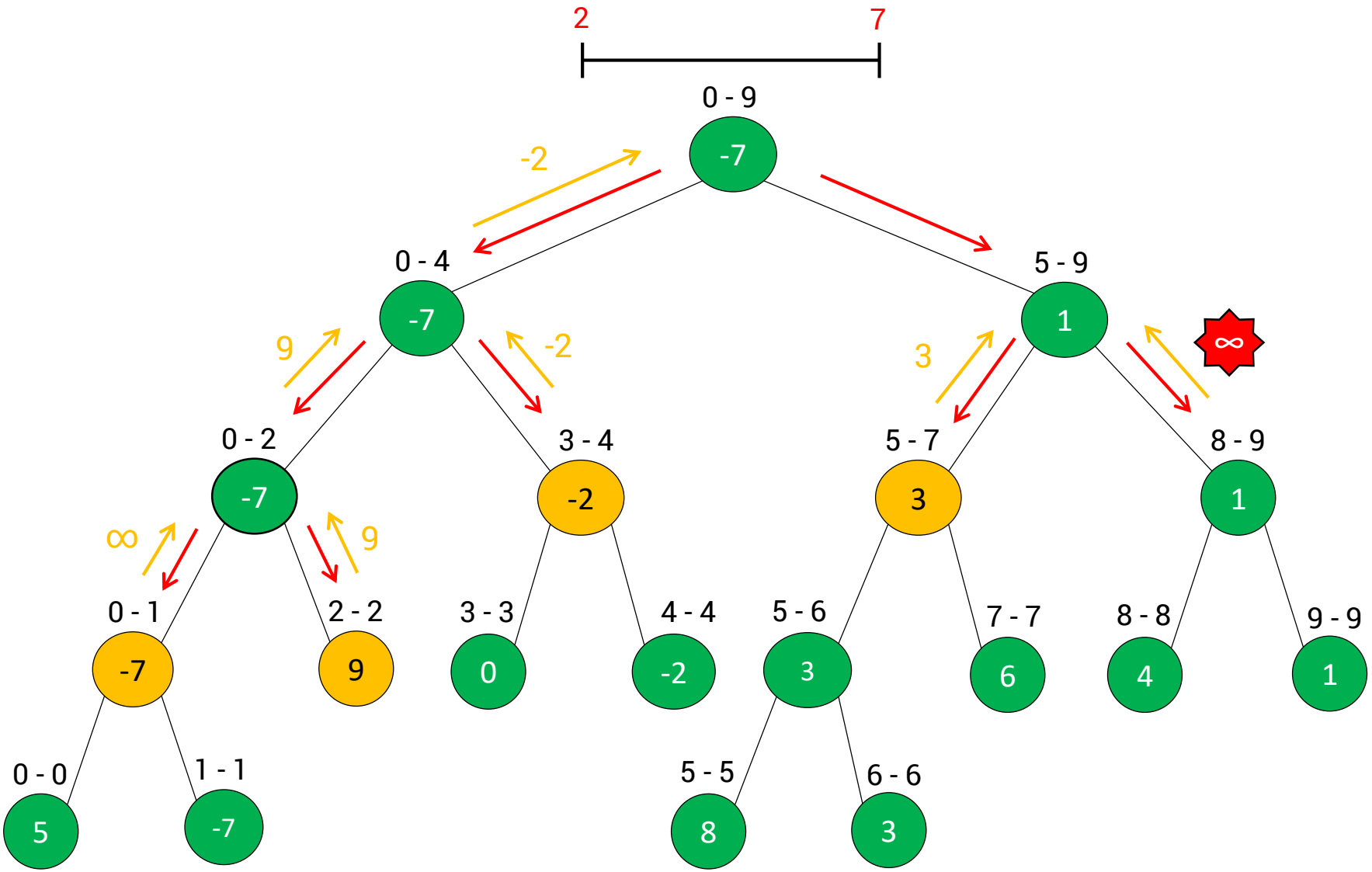
Giá trị nhỏ nhất nhánh trái: -2

Bước 5: Tìm giá trị nhỏ nhất nhánh 4



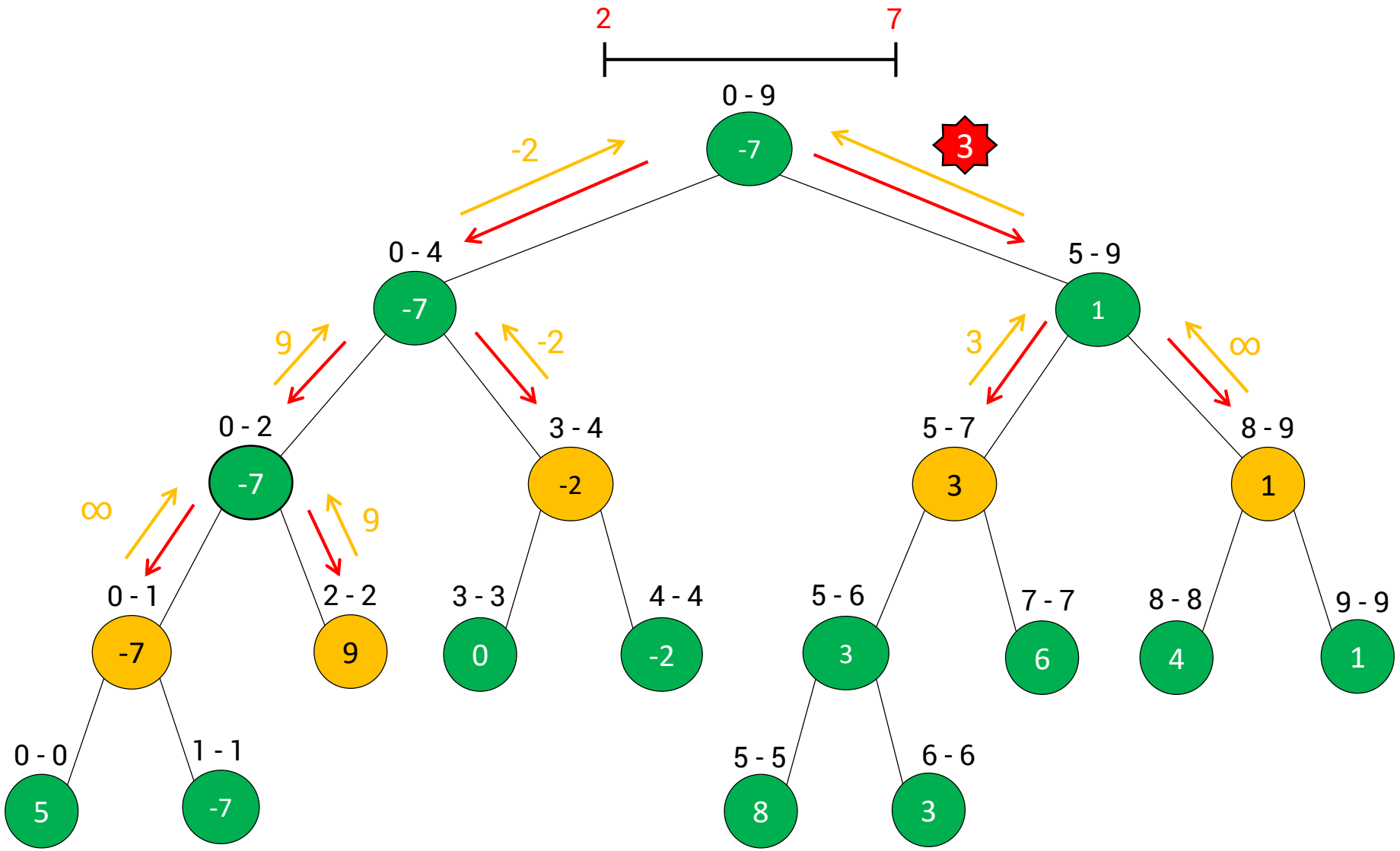
Giá trị nhỏ nhất nhánh 4: 3

Bước 6: Tìm giá trị nhỏ nhất nhánh 5



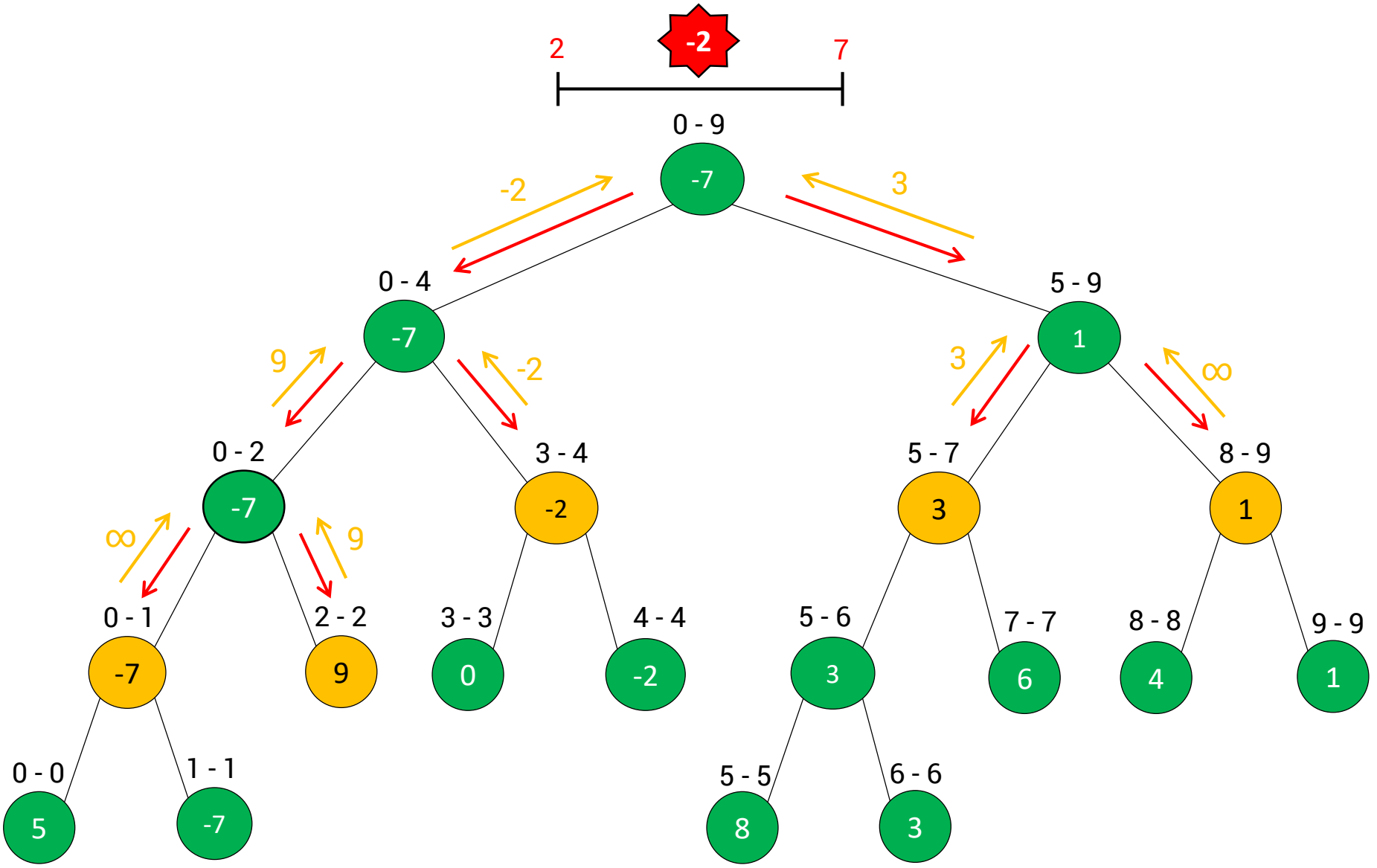
Giá trị nhỏ nhất nhánh 5: ∞

Bước 7: Tìm giá trị nhỏ nhất của nhánh phải



Giá trị nhỏ nhất nhánh phải: 3

Bước 8: Tìm giá trị nhỏ nhất của đoạn [2, 7]



Giá trị nhỏ nhất đoạn [2, 7]: -2

Một số công thức cần nhớ

Để biểu diễn một cây nhị phân đầy đủ bằng mảng bạn cần nhớ một số công thức sau:

1. Chiều cao của cây: $h = [\log(n)]$

2. Số lượng phần tử Segment Tree: $2 * 2\text{^}h - 1$

3. Node con trái của node thứ i: $2 * i + 1$

4. Node con phải của node thứ i: $2 * i + 2$

5. Node con cha của node thứ i: $(i - 1)/2$

Source Code Range Minimum Query

```cpp
int minRange(vector<int> &segtree, int left, int right, int from, int to, int index) {
    if (from <= left && to >= right) {
        return segtree[index];
    }
    if (from > right || to < left) {
        return INF;
    }
    int mid = (left + right) / 2;
    int a = minRange(segtree, left, mid, from, to, 2 * index + 1);
    int b = minRange(segtree, mid + 1, right, from, to, 2 * index + 2);
    return min(a, b);
}
```

Source Code Range Minimum Query

```
13. int main() {
14.     vector<int> a = { 5, -7, 9, 0, -2, 8, 3, 6, 4, 1 };
15.     int n = a.size();
16.     //Height of segment tree
17.     int h = (int)ceil(log2(n));
18.     //Maximum size of segment tree
19.     int sizeTree = 2 * (int)pow(2, h) - 1;
20.     vector<int> segtree(sizeTree, INF);
21.     buildTree(a, segtree, 0, n - 1, 0);
22.     int fromRange = 2;
23.     int toRange = 7;
24.     int min = minRange(segtree, 0, n - 1, fromRange, toRange, 0);
25.     cout << "Range minimum query: " << min << endl;
26.     return 0;
27. }
```

Source Code Range Minimum Query

```python
from math import ceil, log2
INF = 10**9

def minRange(segtree, left, right, fr, to, index):
    if fr <= left and right <= to:
        return segtree[index]
    if fr > right or to < left:
        return INF
    mid = (left + right) // 2
    a = minRange(segtree, left, mid, fr, to, 2 * index + 1)
    b = minRange(segtree, mid + 1, right, fr, to, 2 * index + 2)
    return min(a, b)
```

Source Code Range Minimum Query

```
13. if __name__ == '__main__':
14.     a = [5, -7, 9, 0, -2, 8, 3, 6, 4, 1]
15.     n = len(a)
16.     h = ceil(log2(n))
17.     sizeTree = 2 * (2**h) - 1
18.     segtree = [INF] * sizeTree
19.     buildTree(a, segtree, 0, n - 1, 0)
20.     fromRange = 2
21.     toRange = 7
22.     minValue = minRange(segtree, 0, n - 1, fromRange, toRange, 0)
23.     print(minValue)
```

Source Code Range Minimum Query

```java
import java.util.Arrays;
public class Main {
    private static final int INF = (int)1e9;
    private static double log2(int number) {
        return Math.log(number) / Math.log(2);
    }
    private static int minRange(int[] segtree, int left, int right, int from,
                                int to, int index) {
        if (from <= left && to >= right) {
            return segtree[index];
        }
        if (from > right || to < left) {
            return INF;
        }
        int mid = (left + right) / 2;
        int a = minRange(segtree, left, mid, from, to, 2 * index + 1);
        int b = minRange(segtree, mid + 1, right, from, to, 2 * index + 2);
        return Math.min(a, b);
    }
```

Source Code Range Minimum Query

```
19.     public static void main(String[] args) {
20.         int[] a = new int[]{ 5, -7, 9, 0, -2, 8, 3, 6, 4, 1 };
21.         int n = a.length;
22.         //Height of segment tree
23.         int h = (int)Math.ceil(log2(n));
24.         //Maximum size of segment tree
25.         int sizeTree = 2 * (int)Math.pow(2, h) - 1;
26.         int[] segtree = new int[sizeTree];
27.         Arrays.fill(segtree, INF);
28.         buildTree(a, segtree, 0, n - 1, 0);
29.         int fromRange = 2;
30.         int toRange = 7;
31.         int min = minRange(segtree, 0, n - 1, fromRange, toRange, 0);
32.         System.out.printf("Range minimum query: %d\n", min);
33.     }
```

1. QUERY

SUM OF GIVEN RANGE

Sum of given range

Sum of given range: Cho mảng gồm N phần tử và Q truy vấn. Với mỗi truy vấn, hãy tính tổng giá trị của một đoạn bất kì trên dãy số.

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1

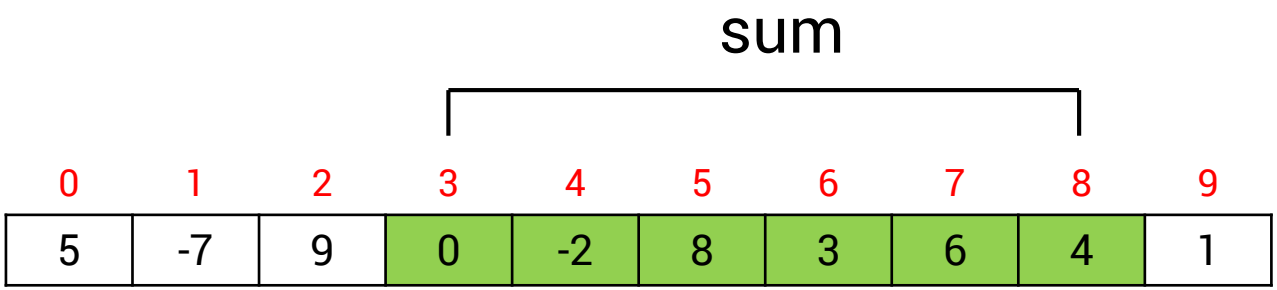
- Truy vấn [3, 8]: 19
- Truy vấn [0, 4]: 5
- Truy vấn [0, 9]: 27
- ...

Các phương pháp giải quyết:

- Nếu dùng Brute Force: **Time complexity** **O(Q*N)**
- Nếu dùng Segment Tree: **Time complexity** **O(N + Q*logN)**

Sum of given range

Chạy truy vấn [3, 8] tính tổng giá trị trên đoạn này.



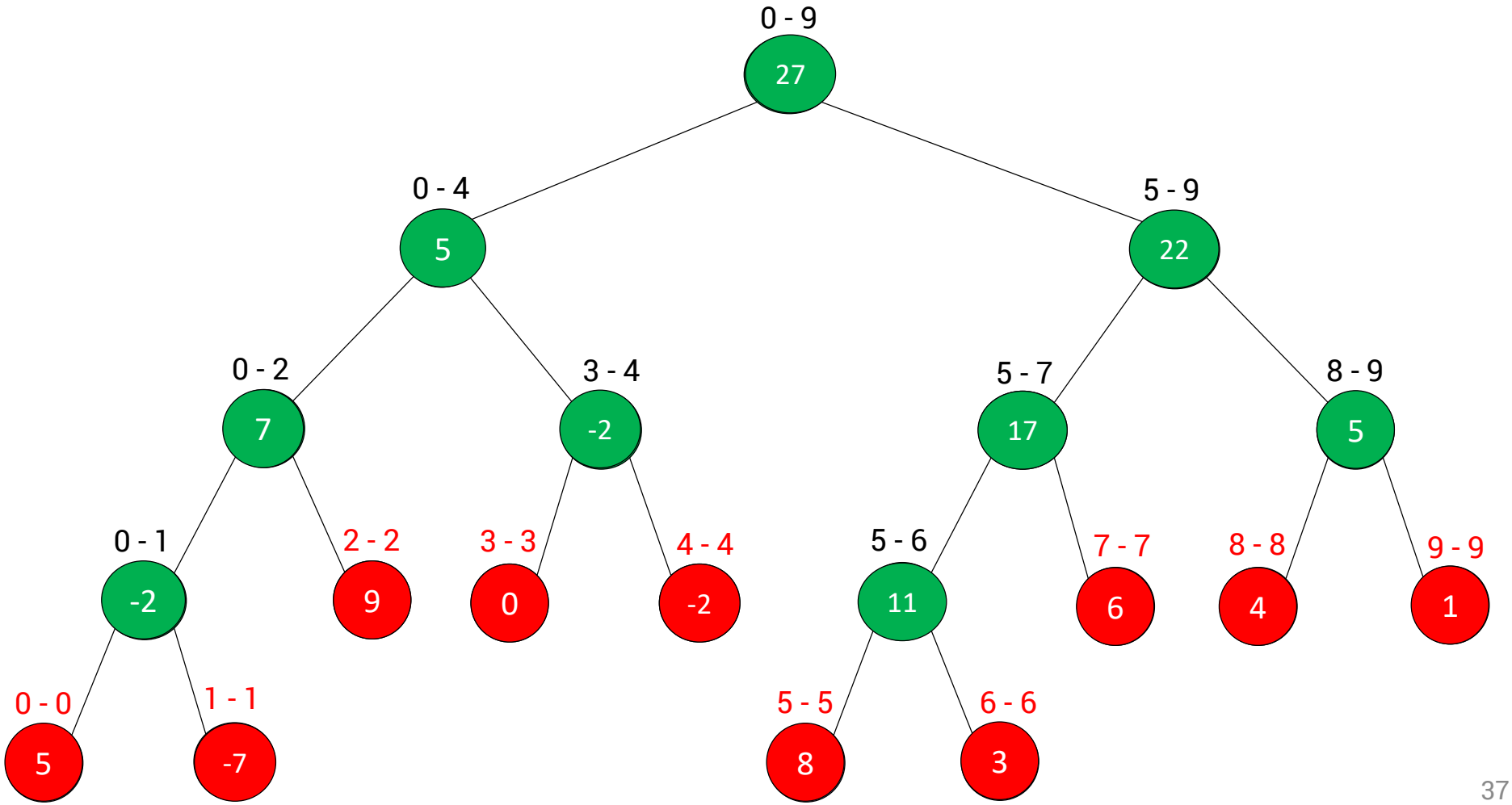
Tổng giá trị đoạn [3, 8]:

sum = 0 + (-2) + 8 + 3 + 6 + 4 = 19.

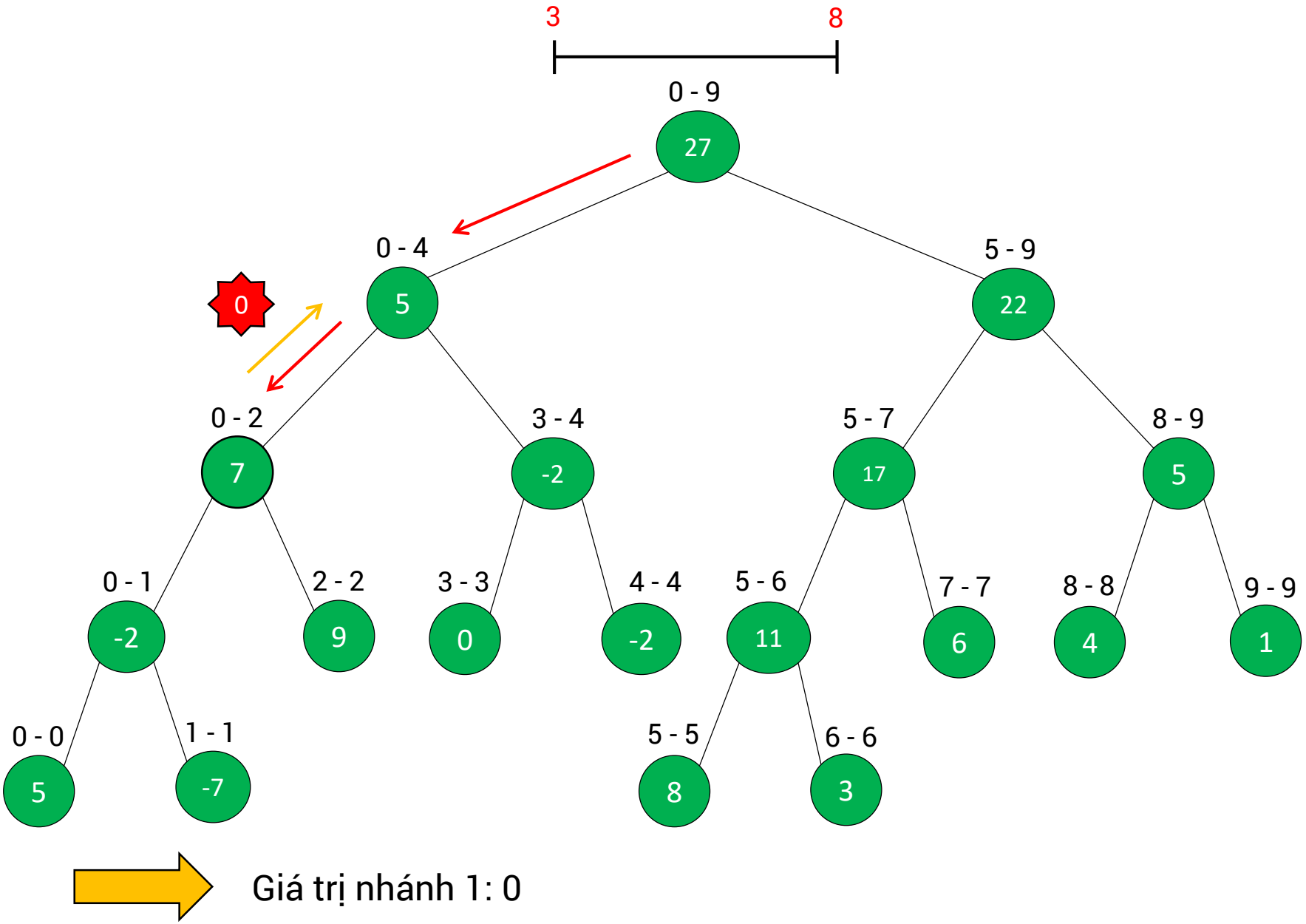
Bước 0: Xây dựng Segment Tree

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1

Tạo cây và điền các giá trị vào Segment Tree:

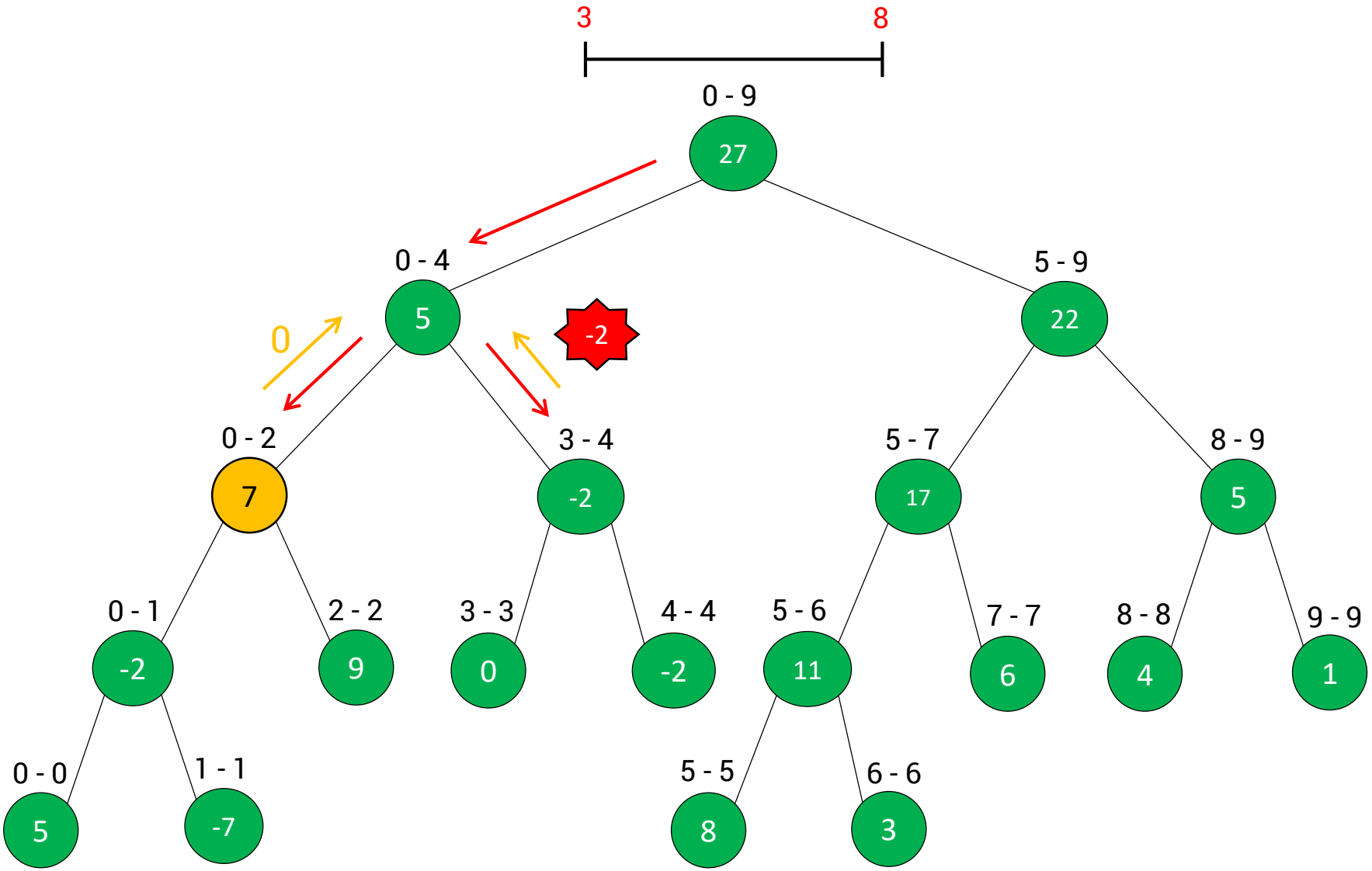


Bước 1: Tìm giá trị nhánh 1



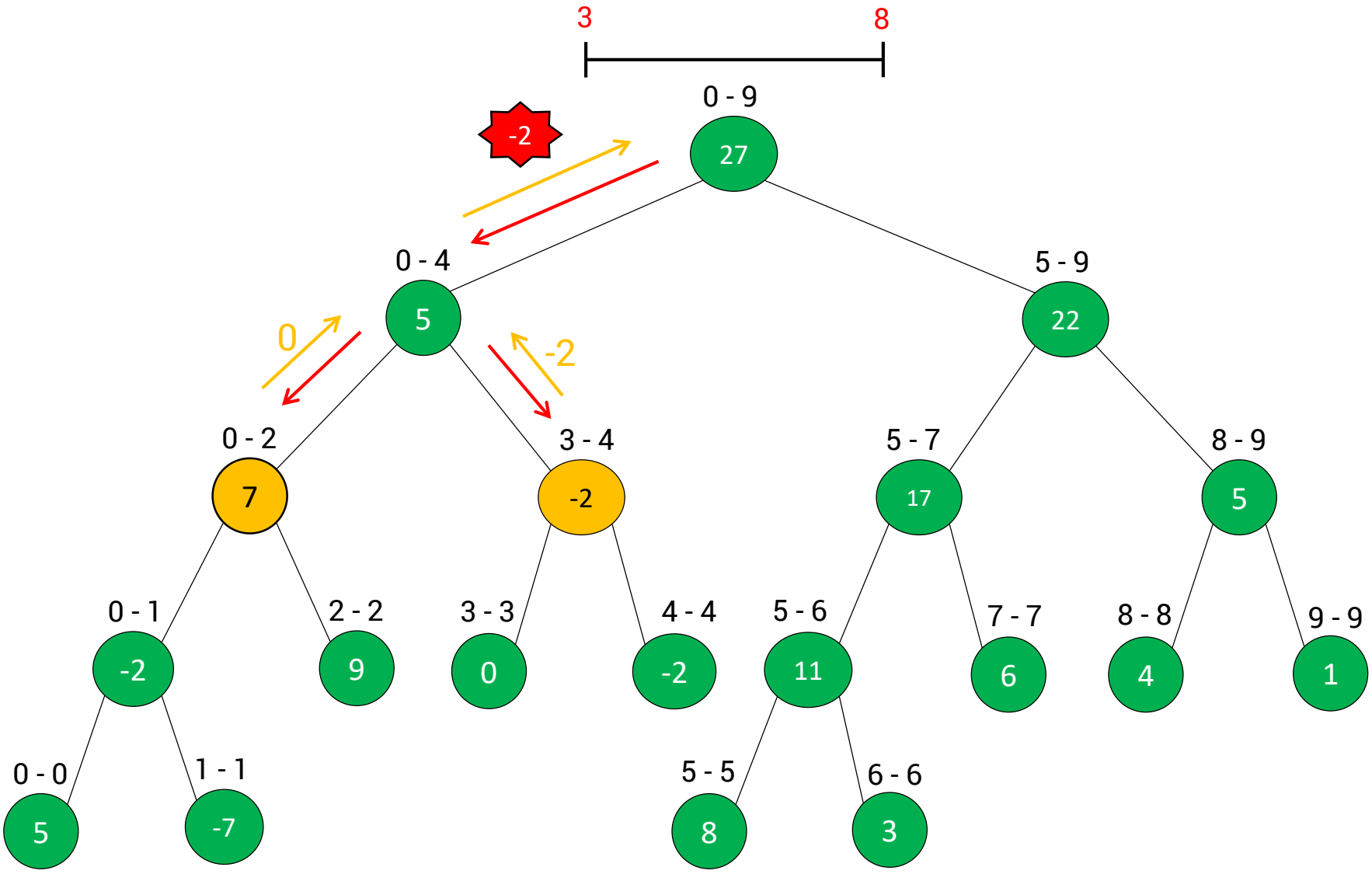
Giá trị nhánh 1: 0

Bước 2: Tìm giá trị nhánh 2



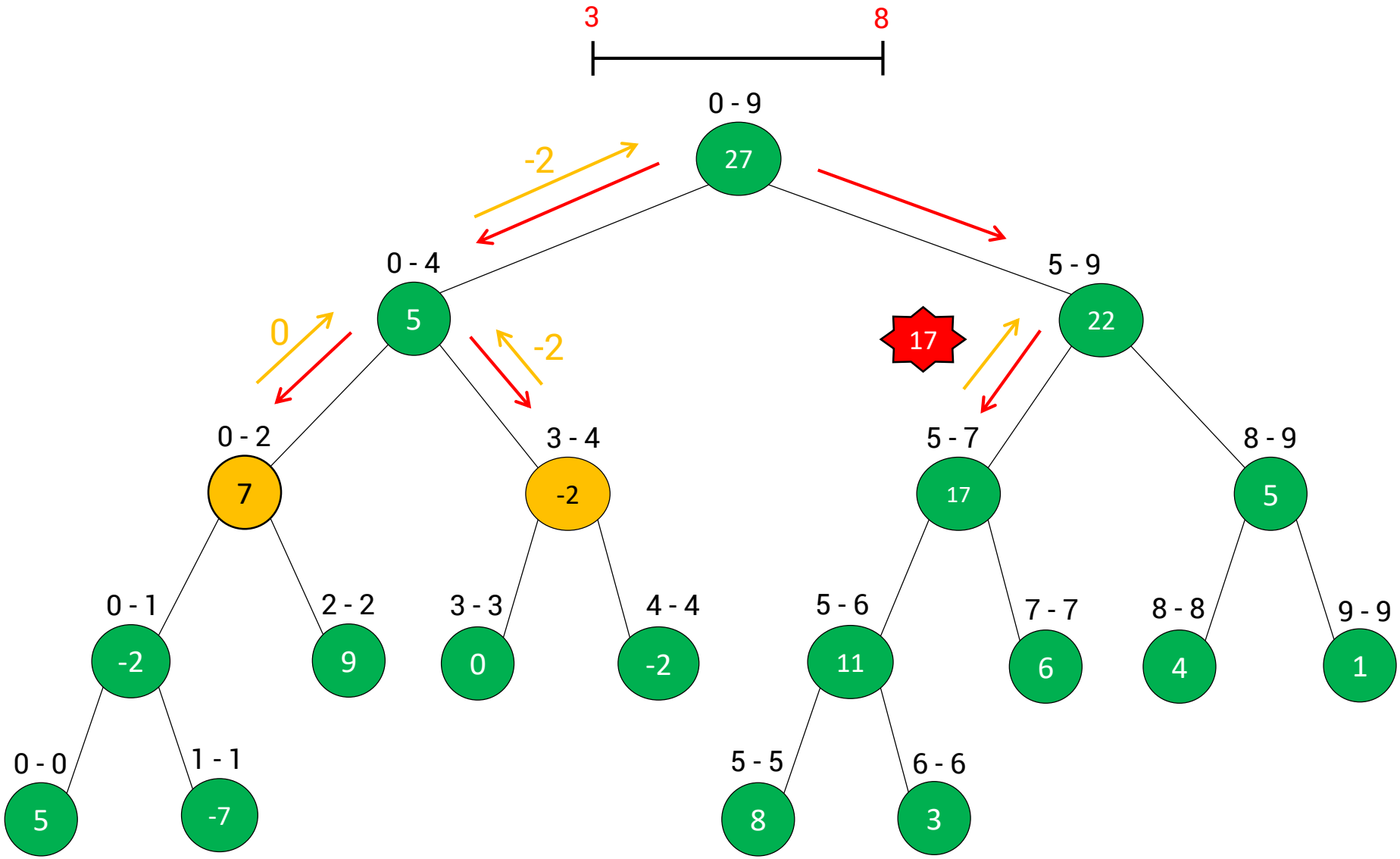
Giá trị nhánh 2: -2

Bước 3: Tìm giá trị toàn bộ nhánh trái



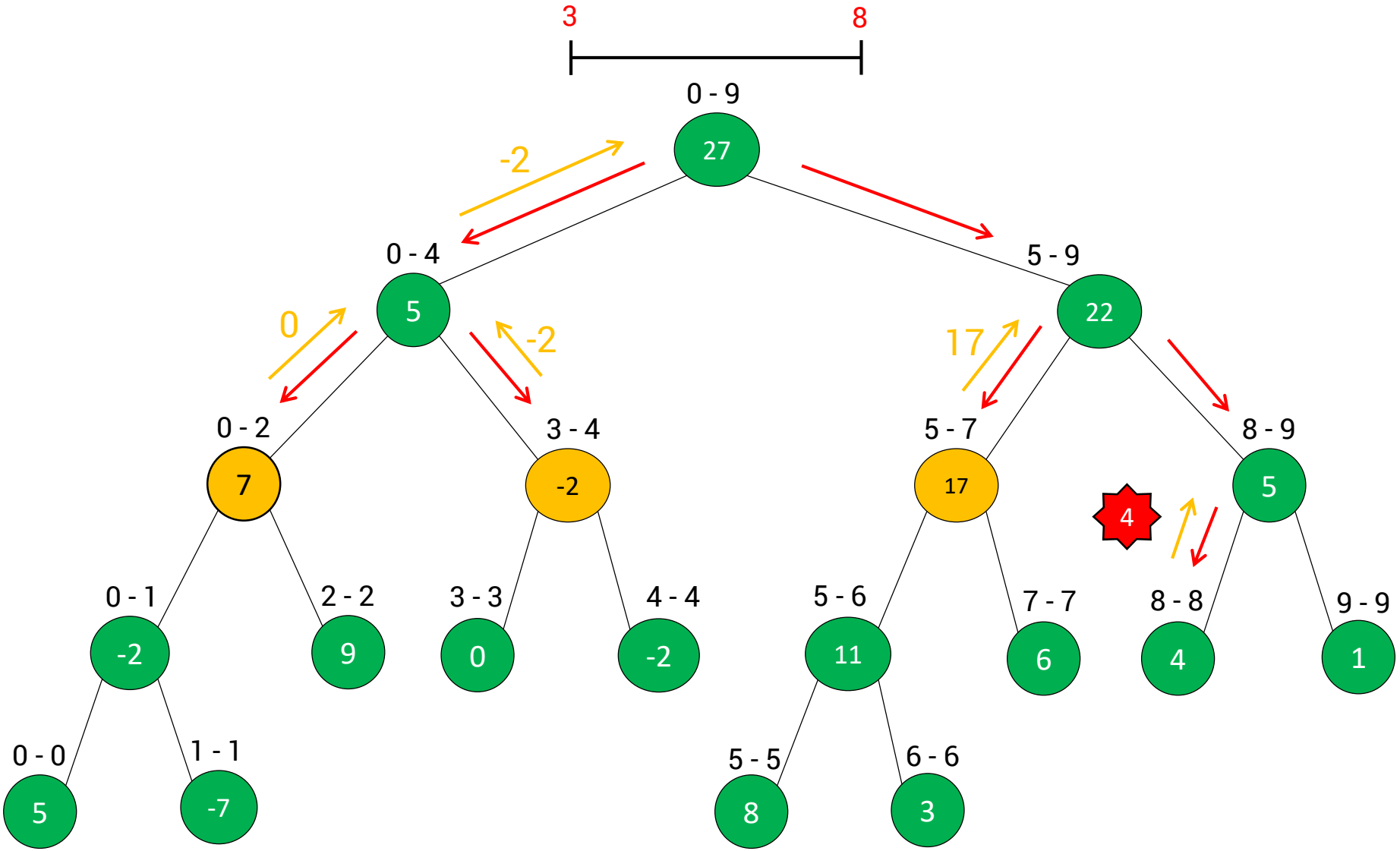
Giá trị toàn bộ nhánh trái: -2

Bước 4: Tìm giá trị nhánh 3



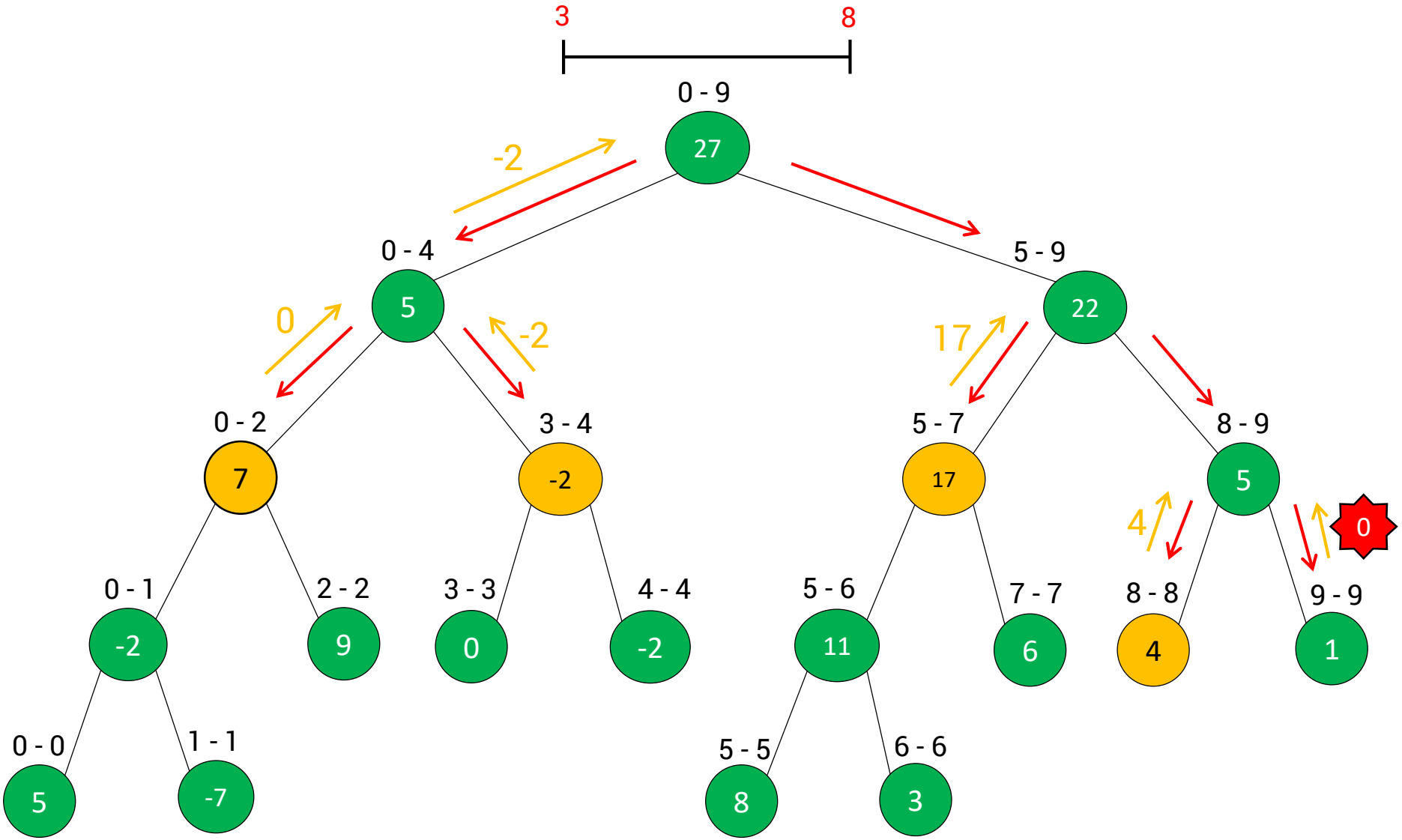
Giá trị nhánh 3: 17

Bước 5: Tìm giá trị nhánh 4



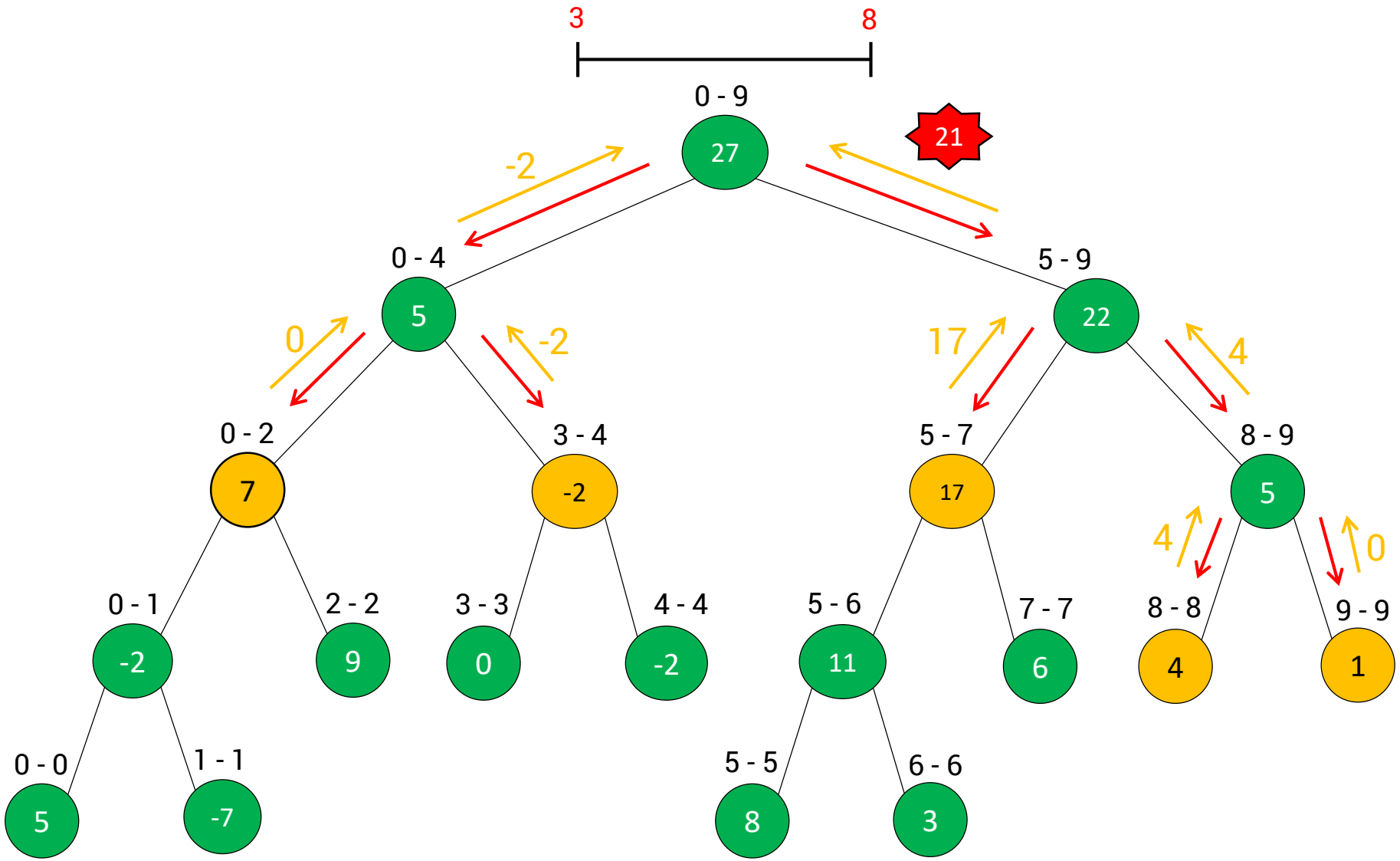
Giá trị nhánh 4: 4

Bước 5: Tìm giá trị nhánh 5



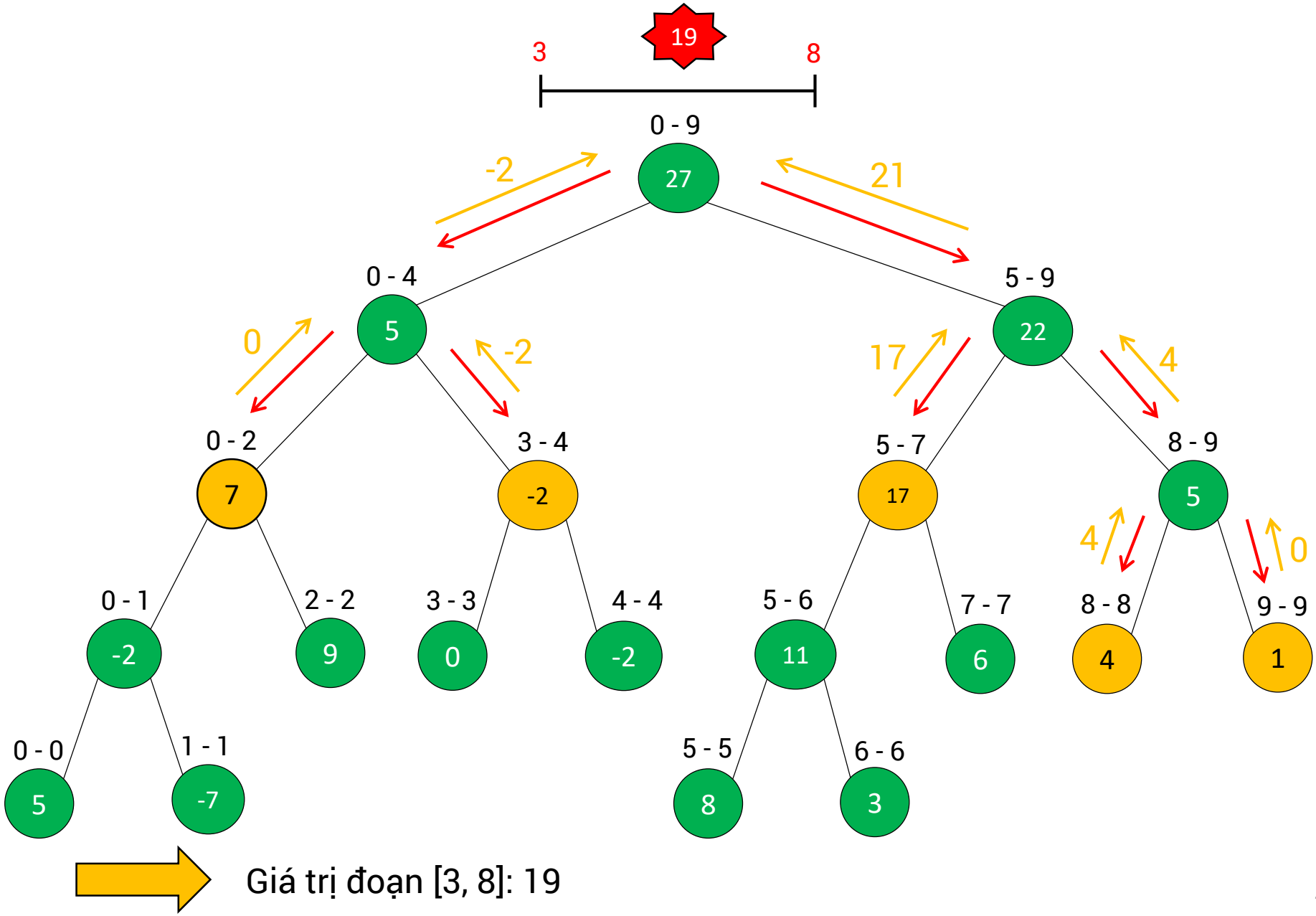
Giá trị nhánh 5: 0

Bước 6: Tìm giá trị nhánh toàn bộ nhánh phải



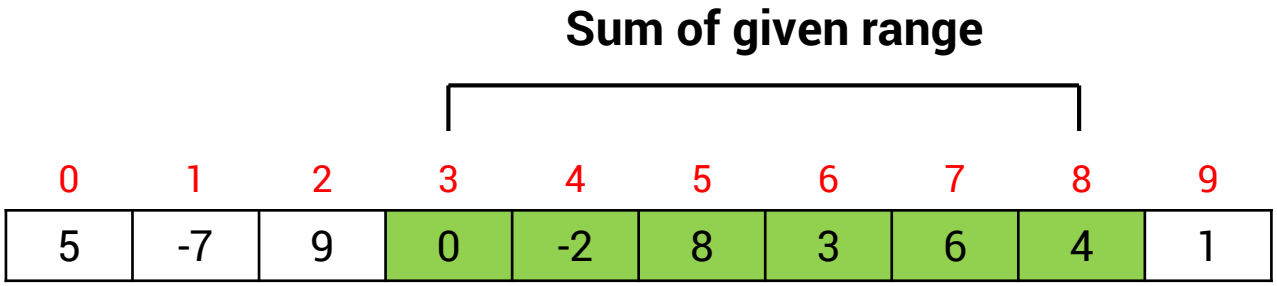
Giá trị toàn bộ nhánh phải: 21

Bước 7: Tổng giá trị đoạn [3, 8]



Giá trị đoạn [3, 8]: 19

Kết quả bài toán



Tổng giá trị đoạn [3, 8]:

Sum = 0 + (-2) + 8 + 3 + 6 + 4 = 19.

Source Code Sum of given range


```cpp
#include <iostream>
#include <algorithm>
#include <cmath>
#include <vector>
using namespace std;
#define INF 1e9
void buildTree(vector<int> &a, vector<int> &segtree, int left, int right, int index) {

    if (left == right) {
        segtree[index] = a[left];
        return;
    }
    int mid = (left + right) / 2;
    buildTree(a, segtree, left, mid, 2 * index + 1);
    buildTree(a, segtree, mid + 1, right, 2 * index + 2);
    segtree[index] = segtree[2 * index + 1] + segtree[2 * index + 2];
}
```

Source Code Sum of given range

```
17. int sumRange(vector<int> &segtree, int left, int right, int from, int to, int index) {
18.     if (from <= left && to >= right) {
19.         return segtree[index];
20.     }
21.     if (from > right || to < left) {
22.         return 0;
23.     }
24.     int mid = (left + right) / 2;
25.     return sumRange(segtree, left, mid, from, to, 2 * index + 1)
26.         + sumRange(segtree, mid + 1, right, from, to, 2 * index + 2);
27. }
```

Source Code Sum of given range

```
28. int main() {
29.     vector<int> a = { 5, -7, 9, 0, -2, 8, 3, 6, 4, 1 };
30.     int n = a.size();
31.     //Height of segment tree
32.     int h = (int)ceil(log2(n));
33.     //Maximum size of segment tree
34.     int sizeTree = 2 * (int)pow(2, h) - 1;
35.     vector<int> segtree(sizeTree);
36.     buildTree(a, segtree, 0, n - 1, 0);
37.     int fromRange = 3;
38.     int toRange = 8;
39.     int sum = sumRange(segtree, 0, n - 1, fromRange, toRange, 0);
40.     cout << "Sum of given range: " << sum << endl;
41.     return 0;
42. }
```

Source Code Sum of given range

```python
from math import ceil, log2
INF = 10**9
def sumRange(segtree, left, right, fr, to, index):
    if fr <= left and to >= right:
        return segtree[index]
    if fr > right or to < left:
        return 0
    mid = (left + right) // 2
    return sumRange(segtree, left, mid, fr, to, 2 * index + 1)
                    + sumRange(segtree, mid + 1, right, fr, to, 2 * index + 2)
```

Source Code Sum of given range

```
10. def buildTree(a, segtree, left, right, index):
11.     if left == right:
12.         segtree[index] = a[left]
13.         return
14.     mid = (left + right) // 2
15.     buildTree(a, segtree, left, mid, 2 * index + 1)
16.     buildTree(a, segtree, mid + 1, right, 2 * index + 2)
17.     segtree[index] = segtree[2 * index + 1] + segtree[2 * index + 2]
```

Source Code Sum of given range

```
18. if __name__ == '__main__':
19.     a = [5, -7, 9, 0, -2, 8, 3, 6, 4, 1]
20.     n = len(a)
21.     h = ceil(log2(n))
22.     sizeTree = 2 * (2**h) - 1
23.     segtree = [INF] * sizeTree
24.     lazy = [0] * sizeTree
25.     buildTree(a, segtree, 0, n - 1, 0)
26.     fromRange = 3
27.     toRange = 8
28.     res = sumRange(segtree, 0, n - 1, fromRange, toRange, 0)
29.     print("Sum of given range:", res)
```

Source Code Sum of given range

```java
import java.util.Arrays;
public class Main {
    private static final int INF = (int)1e9;
    private static double log2(int number) {
        return Math.log(number) / Math.log(2);
    }
    private static void buildTree(int[] a, int[] segtree, int left, int right, int index) {
        if (left == right) {
            segtree[index] = a[left];
            return;
        }
        int mid = (left + right) / 2;
        buildTree(a, segtree, left, mid, 2 * index + 1);
        buildTree(a, segtree, mid + 1, right, 2 * index + 2);
        segtree[index] = segtree[2 * index + 1] + segtree[2 * index + 2];
    }
```

Source Code Sum of given range

```
17.     private static int sumRange(int[] segtree, int left, int right, int from,
                                    int to, int index) {
18.         if (from <= left && to >= right) {
19.             return segtree[index];
20.         }
21.         if (from > right || to < left) {
22.             return 0;
23.         }
24.         int mid = (left + right) / 2;
25.         return sumRange(segtree, left, mid, from, to, 2 * index + 1)
26.                 + sumRange(segtree, mid + 1, right, from, to, 2 * index + 2);
27.     }
```

Source Code Sum of given range

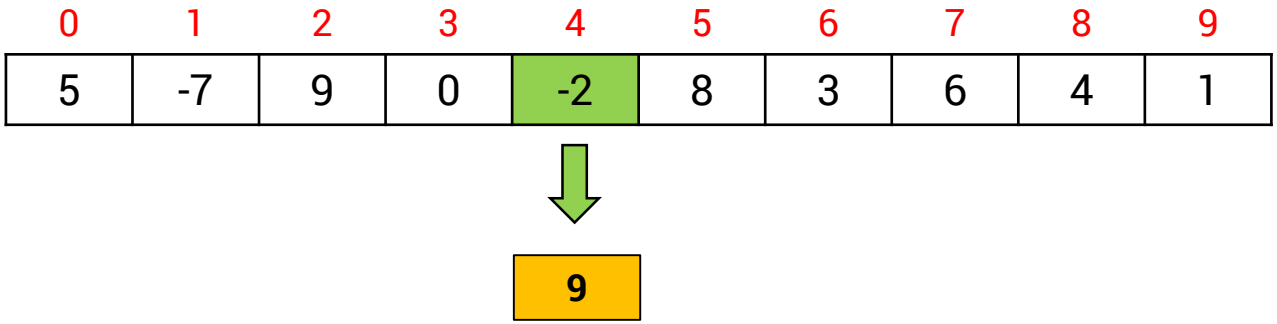
```
28.     public static void main(String[] args) {
29.         int[] a = new int[]{ 5, -7, 9, 0, -2, 8, 3, 6, 4, 1 };
30.         int n = a.length;
31.         //Height of segment tree
32.         int h = (int)Math.ceil(log2(n));
33.         //Maximum size of segment tree
34.         int sizeTree = 2 * (int)Math.pow(2, h) - 1;
35.         int[] segtree = new int[sizeTree];
36.         buildTree(a, segtree, 0, n - 1, 0);
37.         int fromRange = 3;
38.         int toRange = 8;
39.         int min = sumRange(segtree, 0, n - 1, fromRange, toRange, 0);
40.         System.out.printf("Sum of given range: %d\n", min);
41.     }
42. }
```

2. UPDATE

RANGE MINIMUM QUERY

Update: Range Minimum Query

Update RMQ: Thao tác này dùng để cập nhật một giá trị bất kỳ trên Segment Tree, sau đó sử dụng lại truy vấn RMQ thì kết quả sẽ được cập nhật theo giá trị mới nhất trên cây.

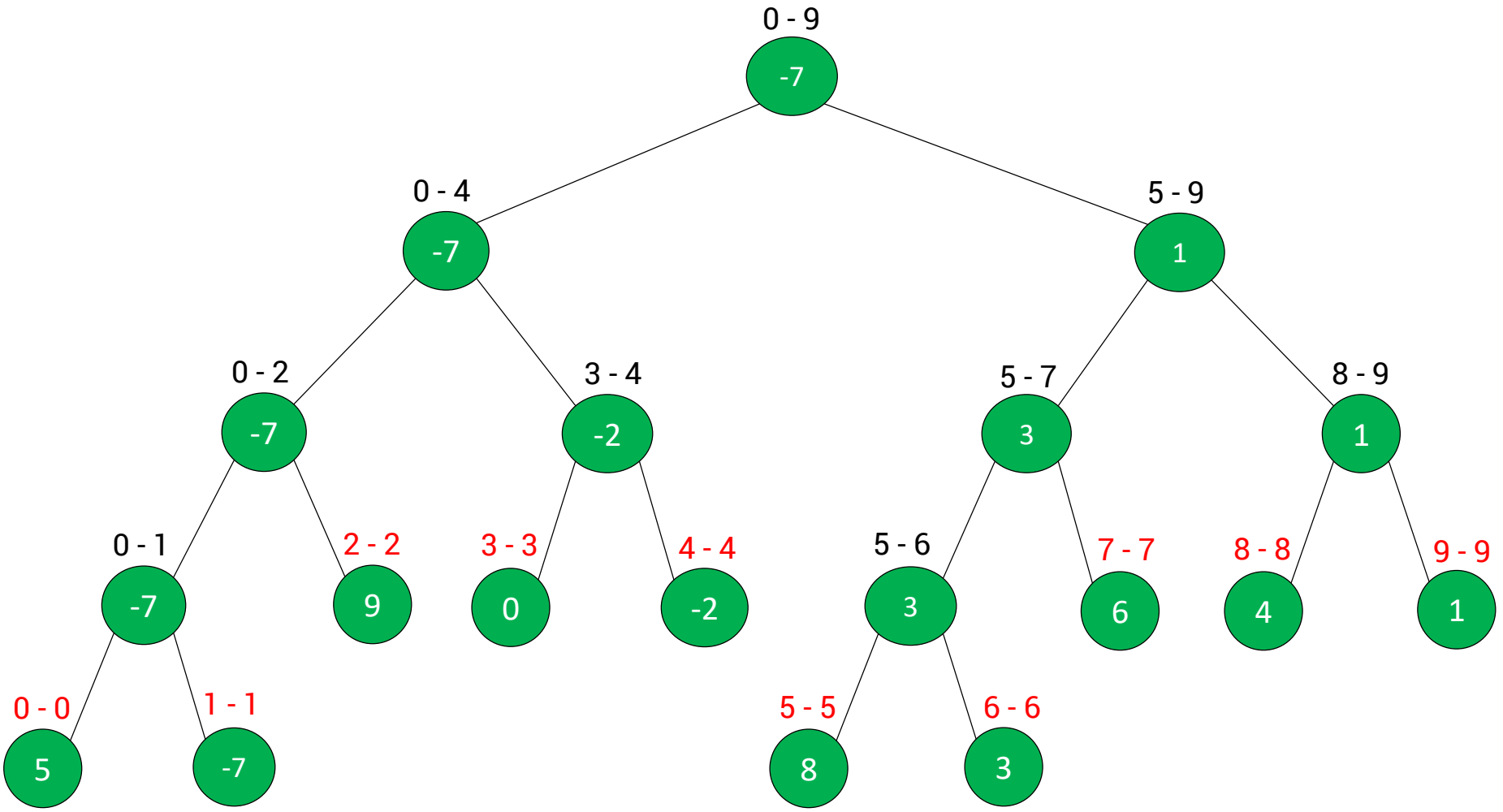


- Truy vấn [2, 7]: -2
- Cập nhật giá trị tại vị trí 4: -2 → 9
- Truy vấn [2, 7]: 0

Time complexity: O(LogN)

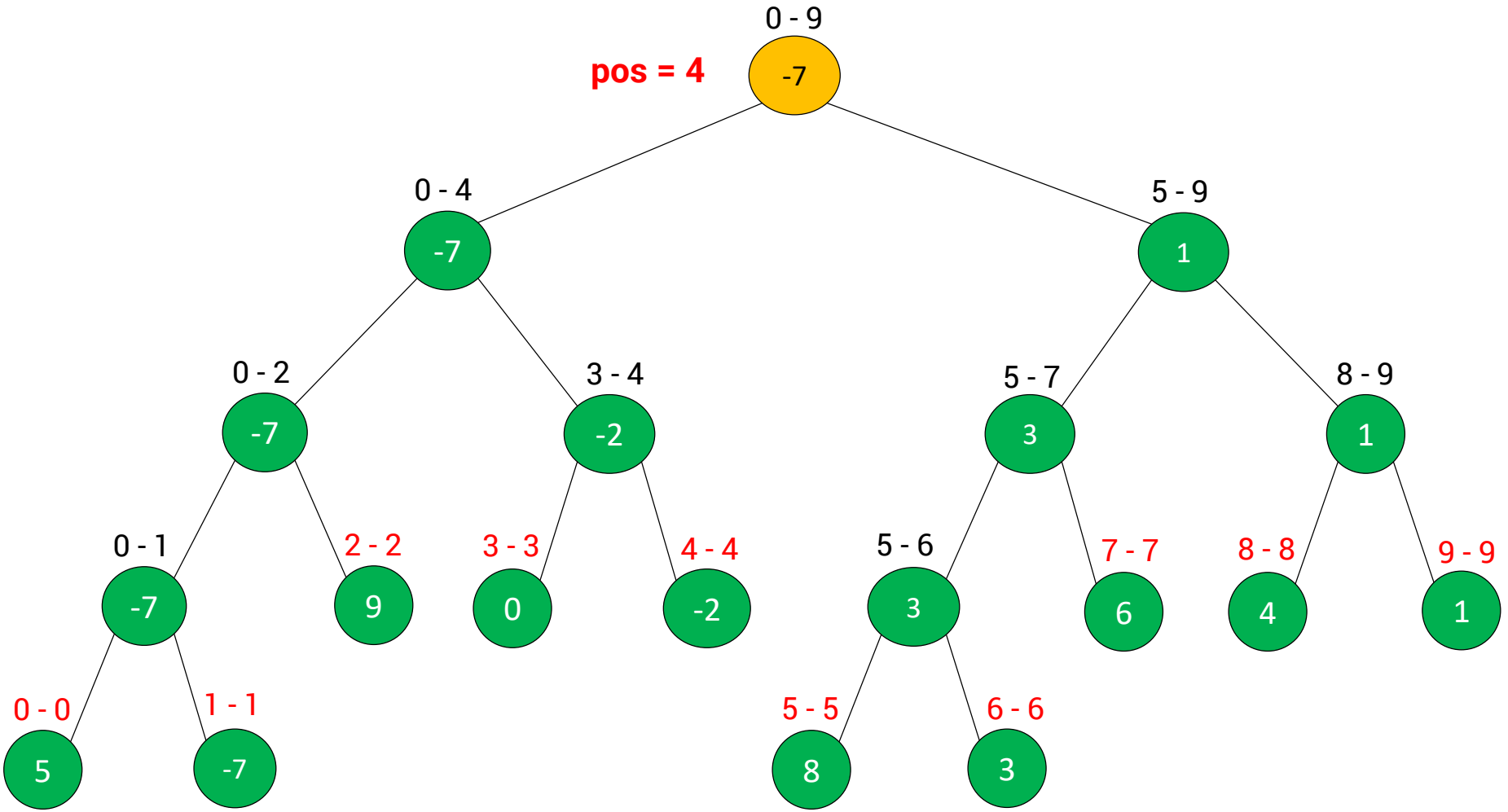
Bước 0: Xây dựng Segment Tree (1)

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1



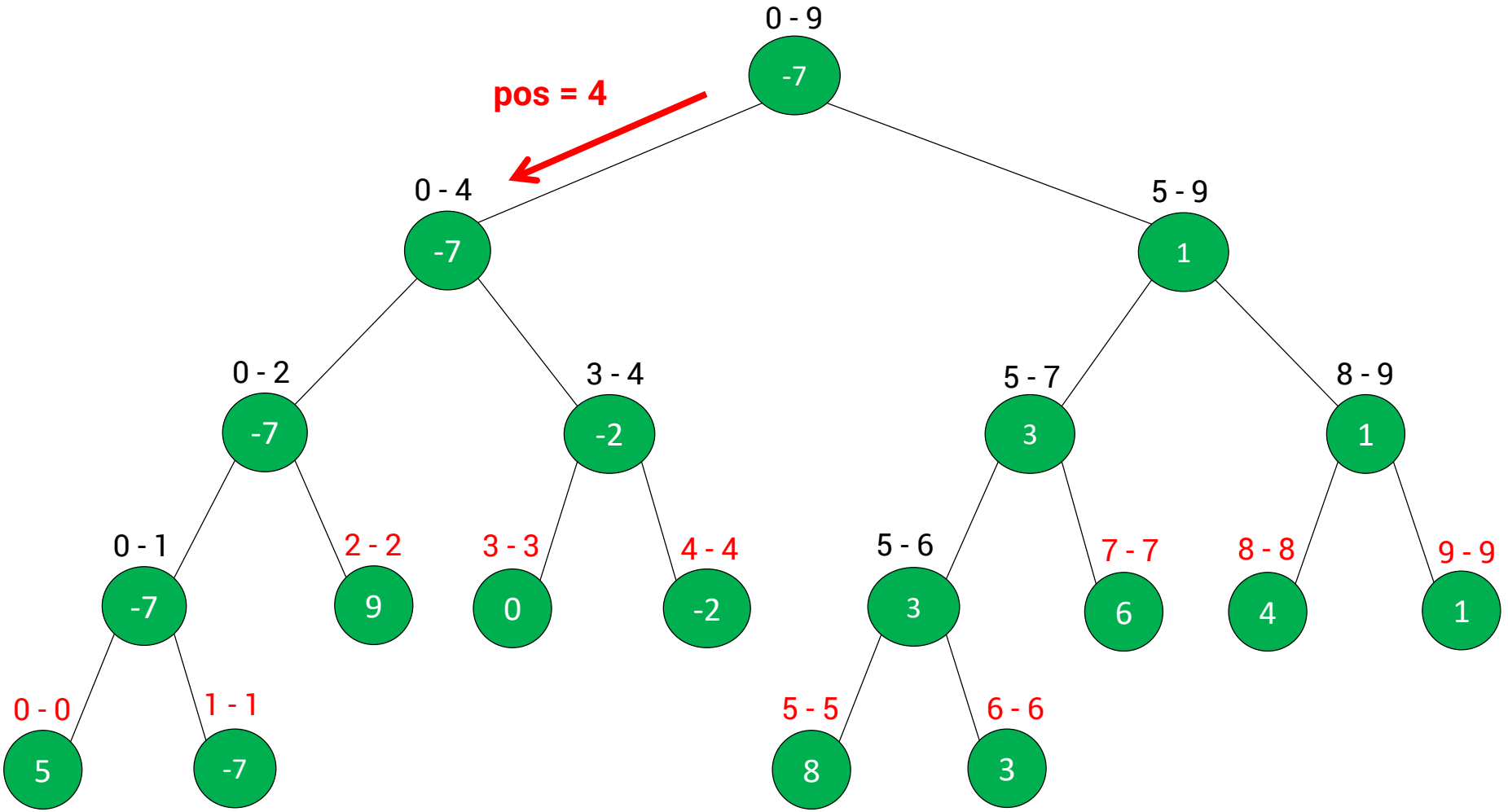
Bước 1: Đi xuống node cần cập nhật

Từ vị trí node gốc xét xem **position** cần cập nhật sẽ thuộc nhánh nào.



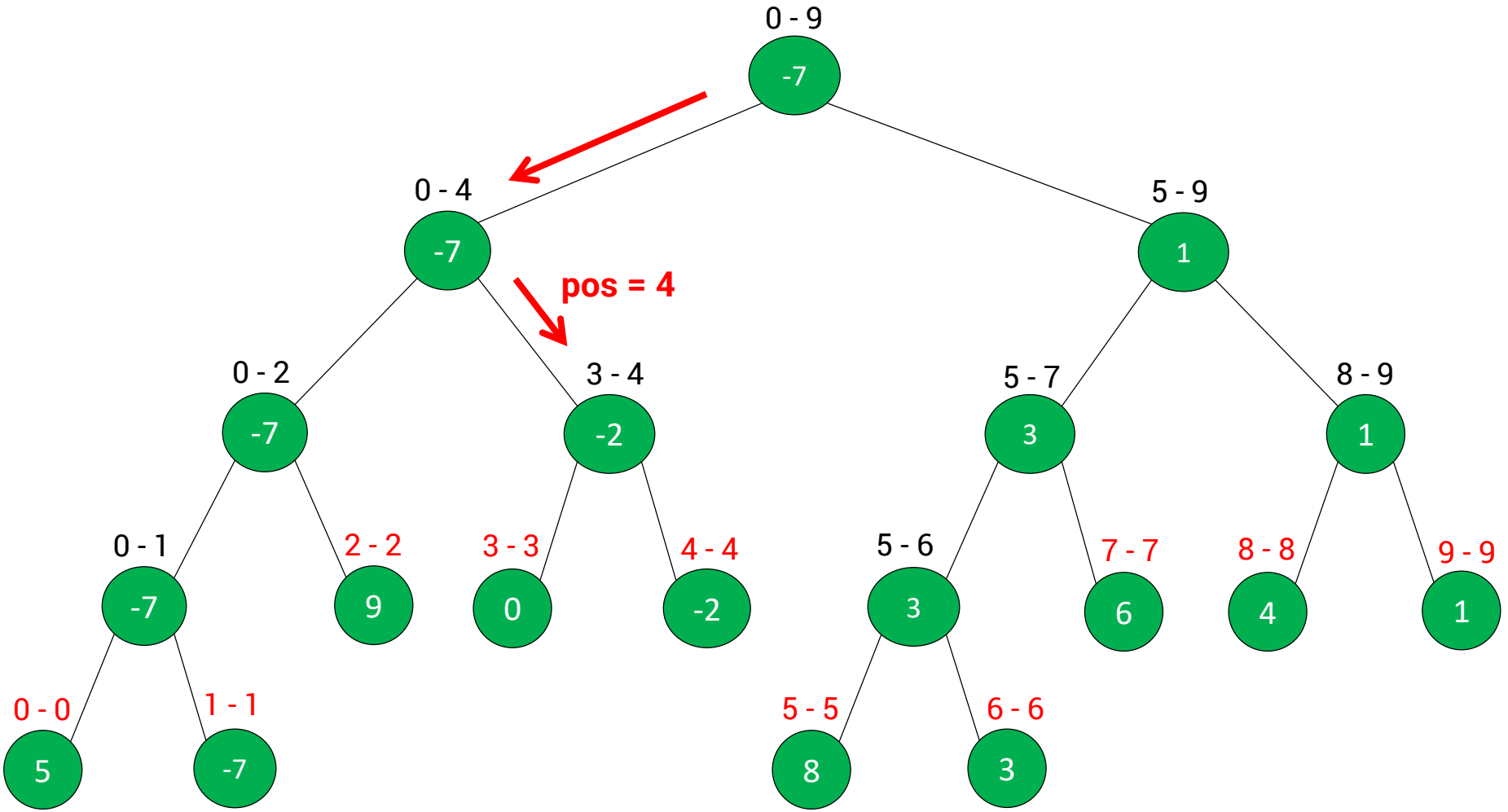
Bước 2: Đi về hướng trái

mid = (left + right) / 2 = (0 + 9) / 2 = 4 = pos(4) ➔ Đi về nhánh trái.



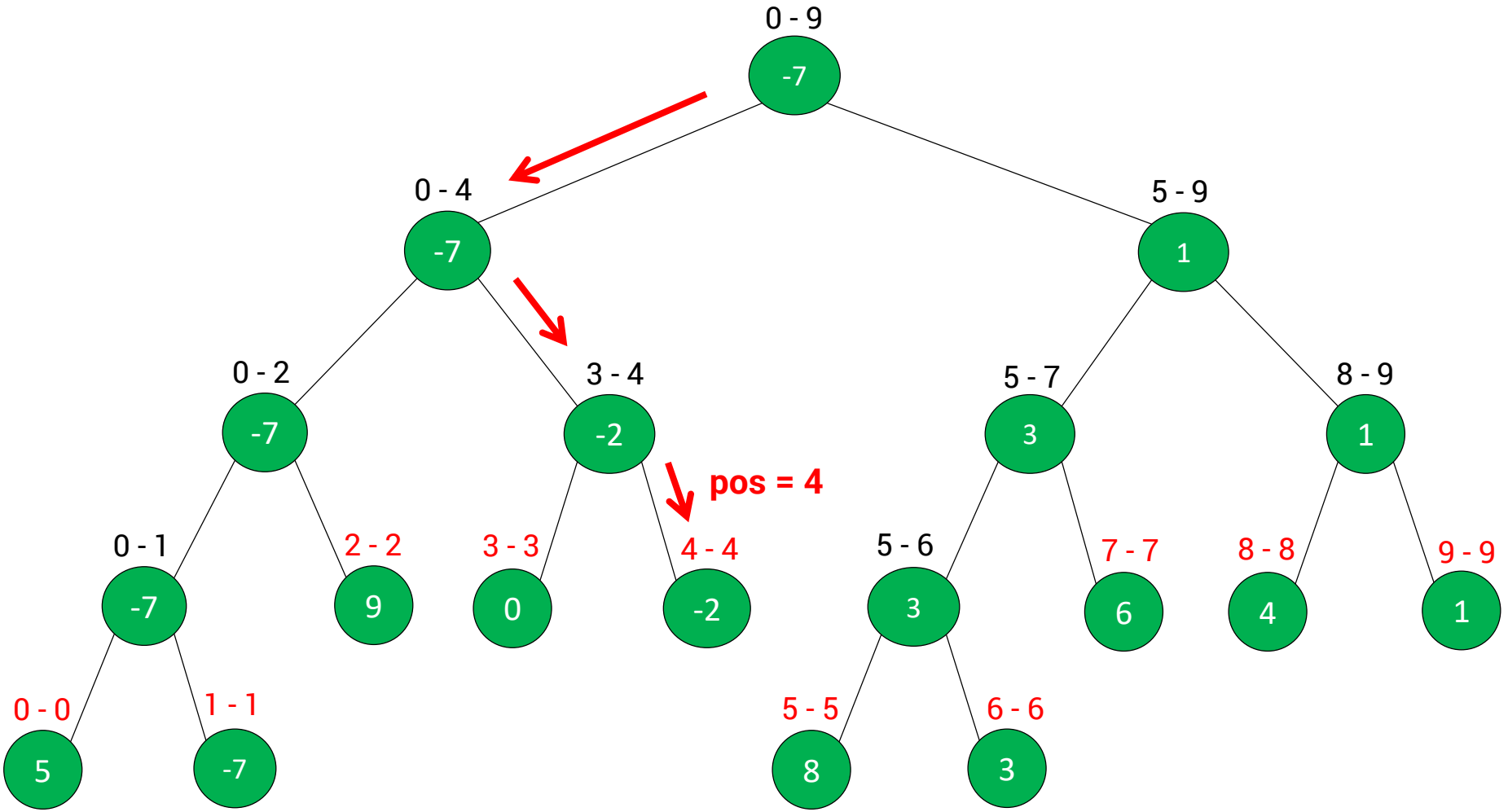
Bước 3: Đi về hướng phải

mid = (left + right) / 2 = (0 + 4) / 2 = 2 < pos(4) ➔ Đi về nhánh phải.



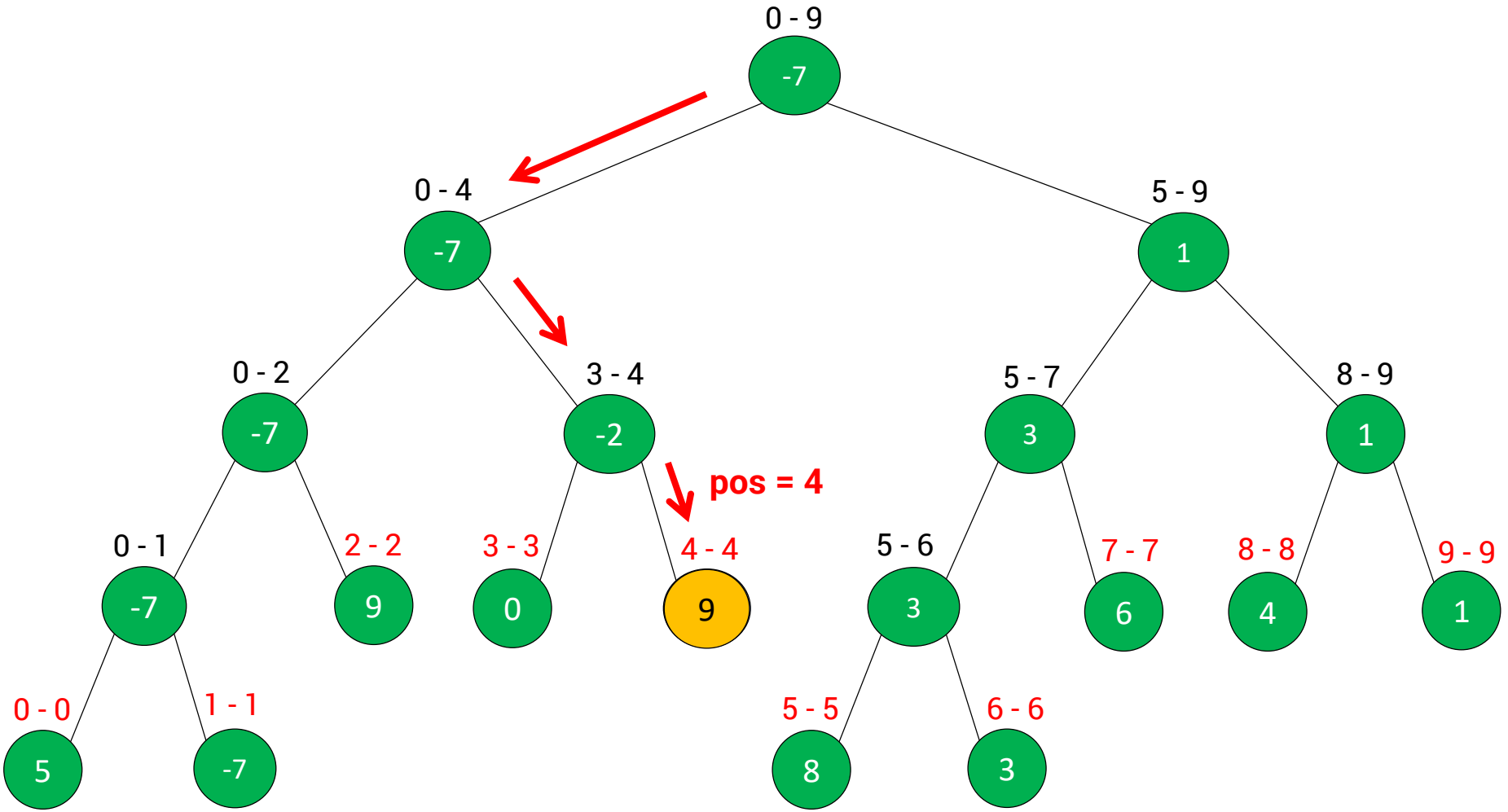
Bước 4: Đi về hướng phải

mid = (left + right) / 2 = (3 + 4) / 2 = 3 < pos(4) ➔ Đi về nhánh phải.



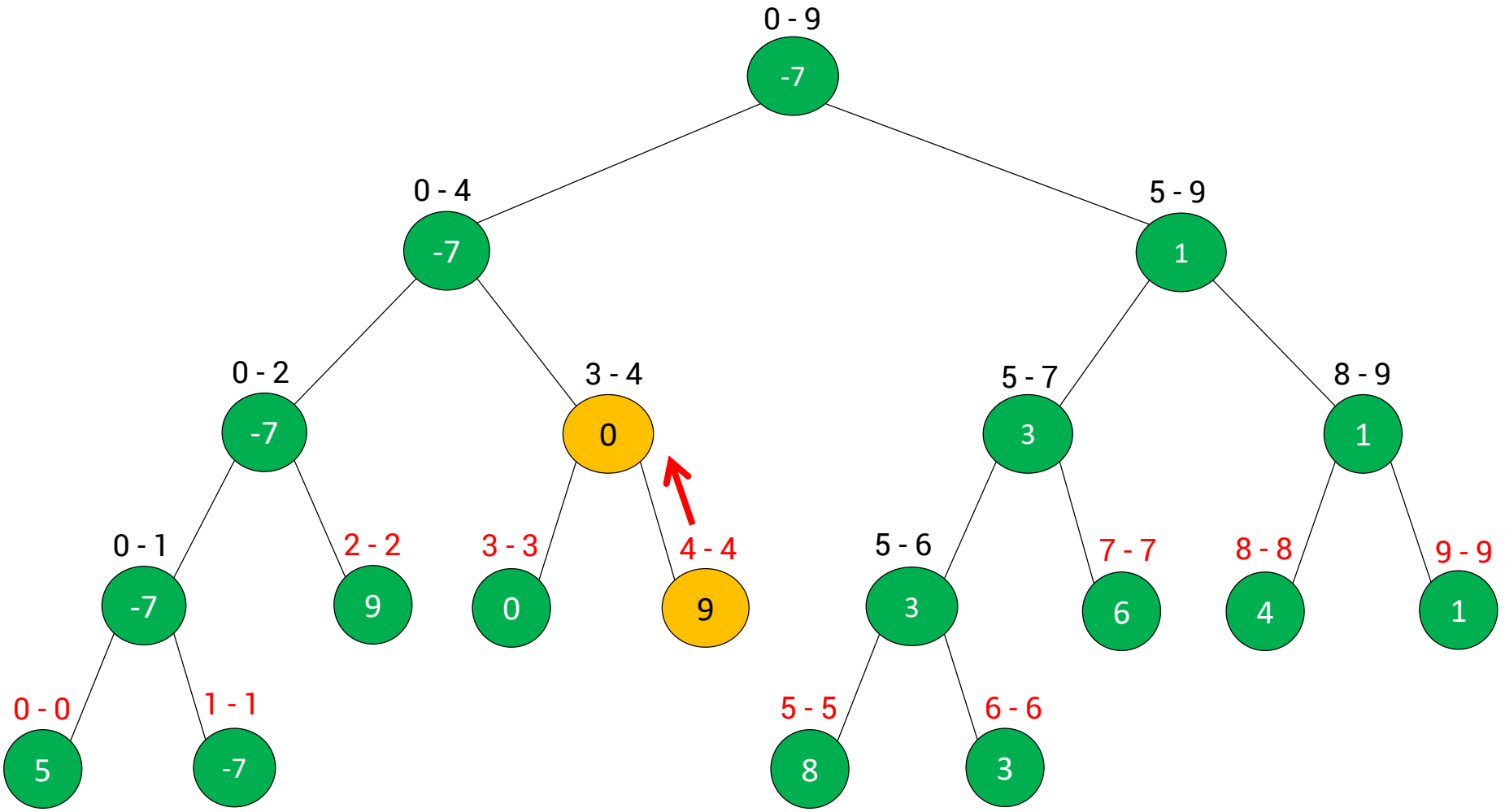
Bước 5: Cập nhật giá trị [4, 4]

Thay đổi giá trị [4 – 4] trên Segment Tree = 9.



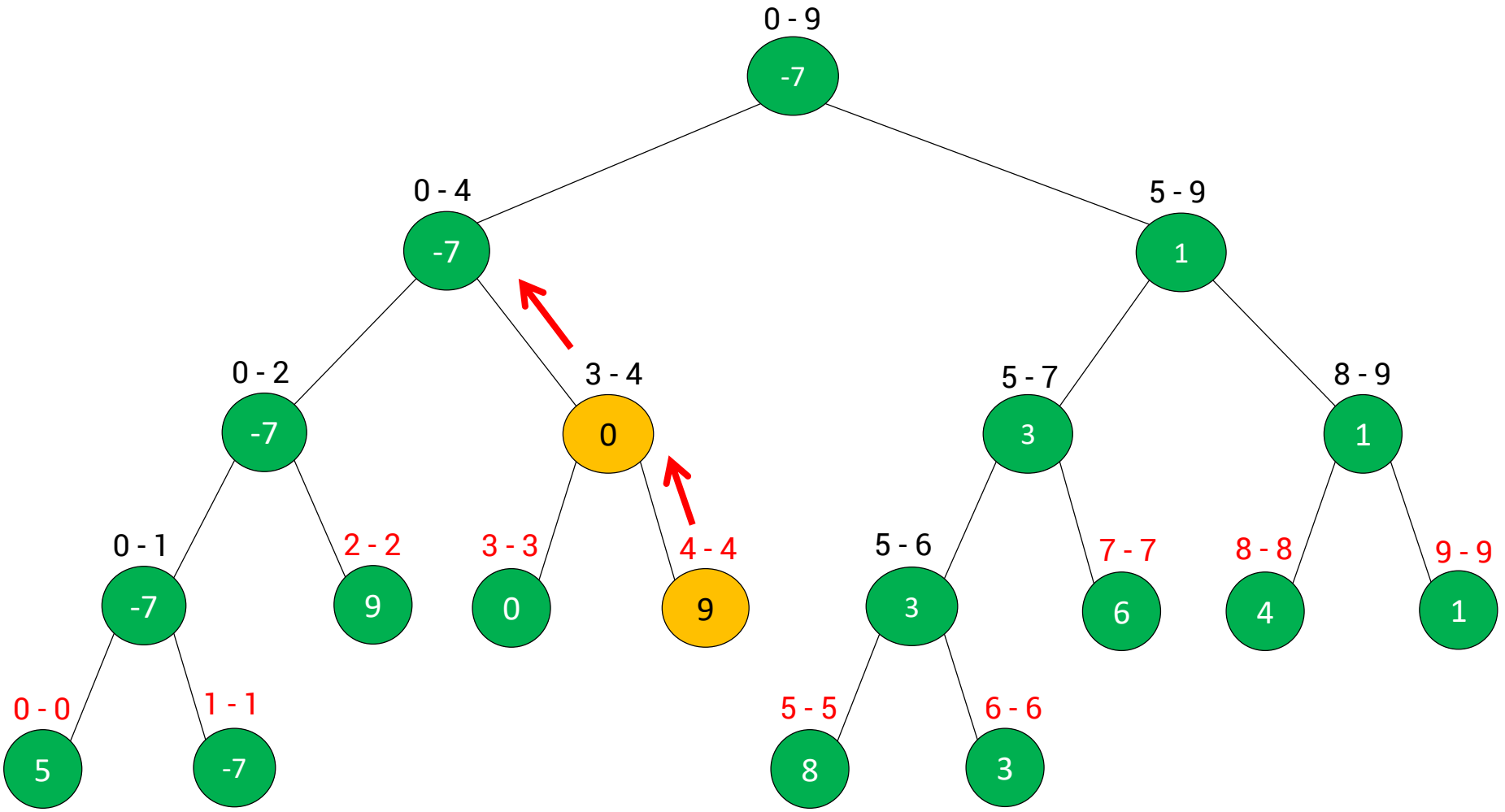
Bước 6: Cập nhật giá trị [3, 4]

Thay đổi giá trị [3 – 4] trên Segment = 0.



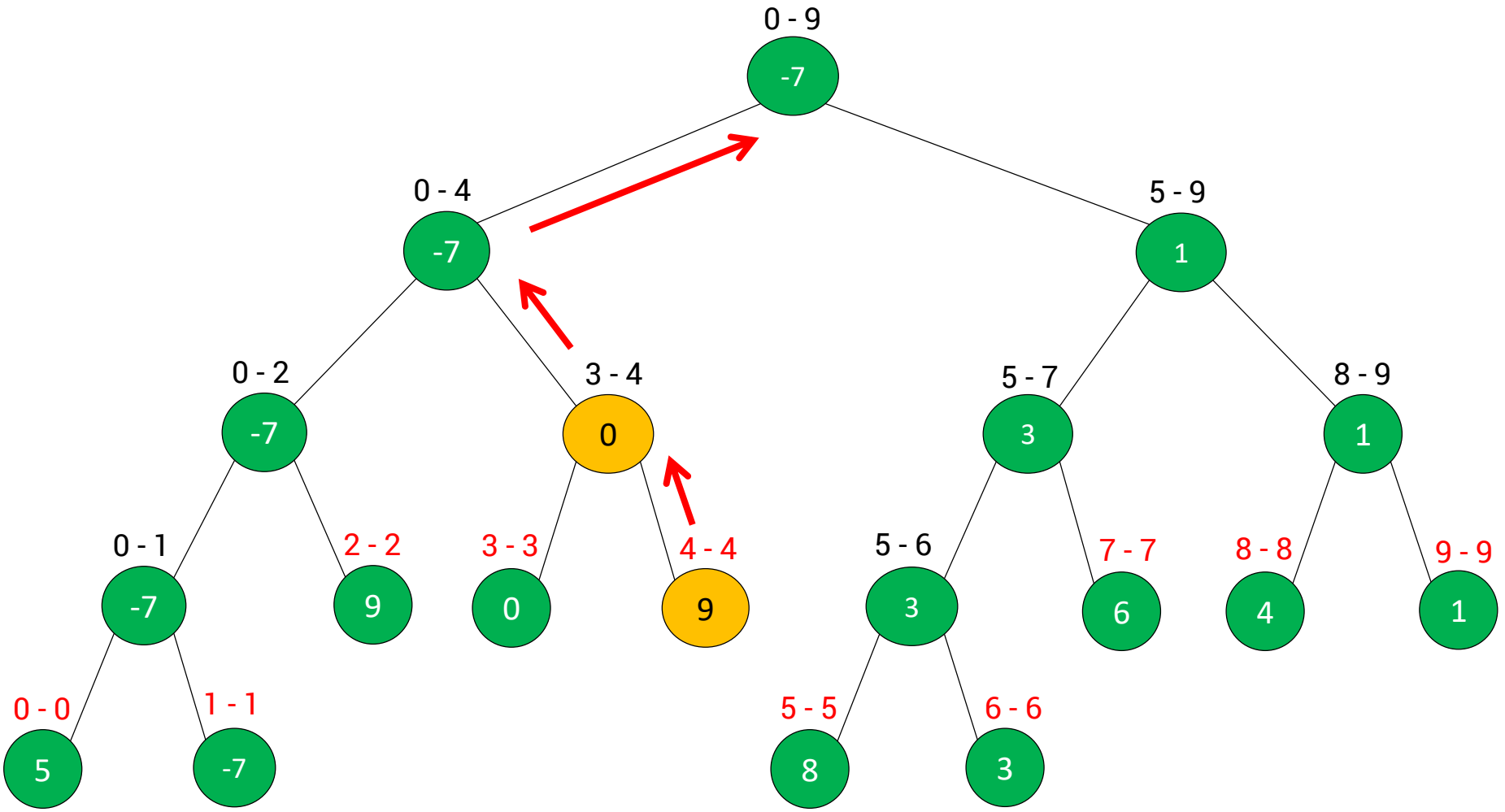
Bước 7: Không cập nhật giá trị [0, 4]

Không thay đổi giá trị [0 – 4] trên Segment.

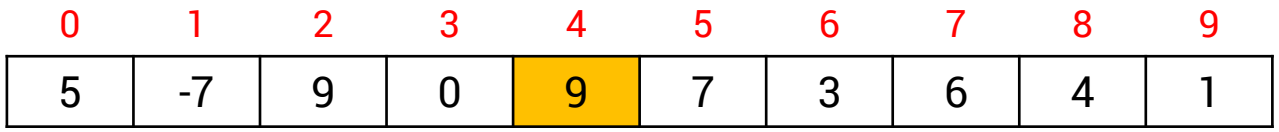


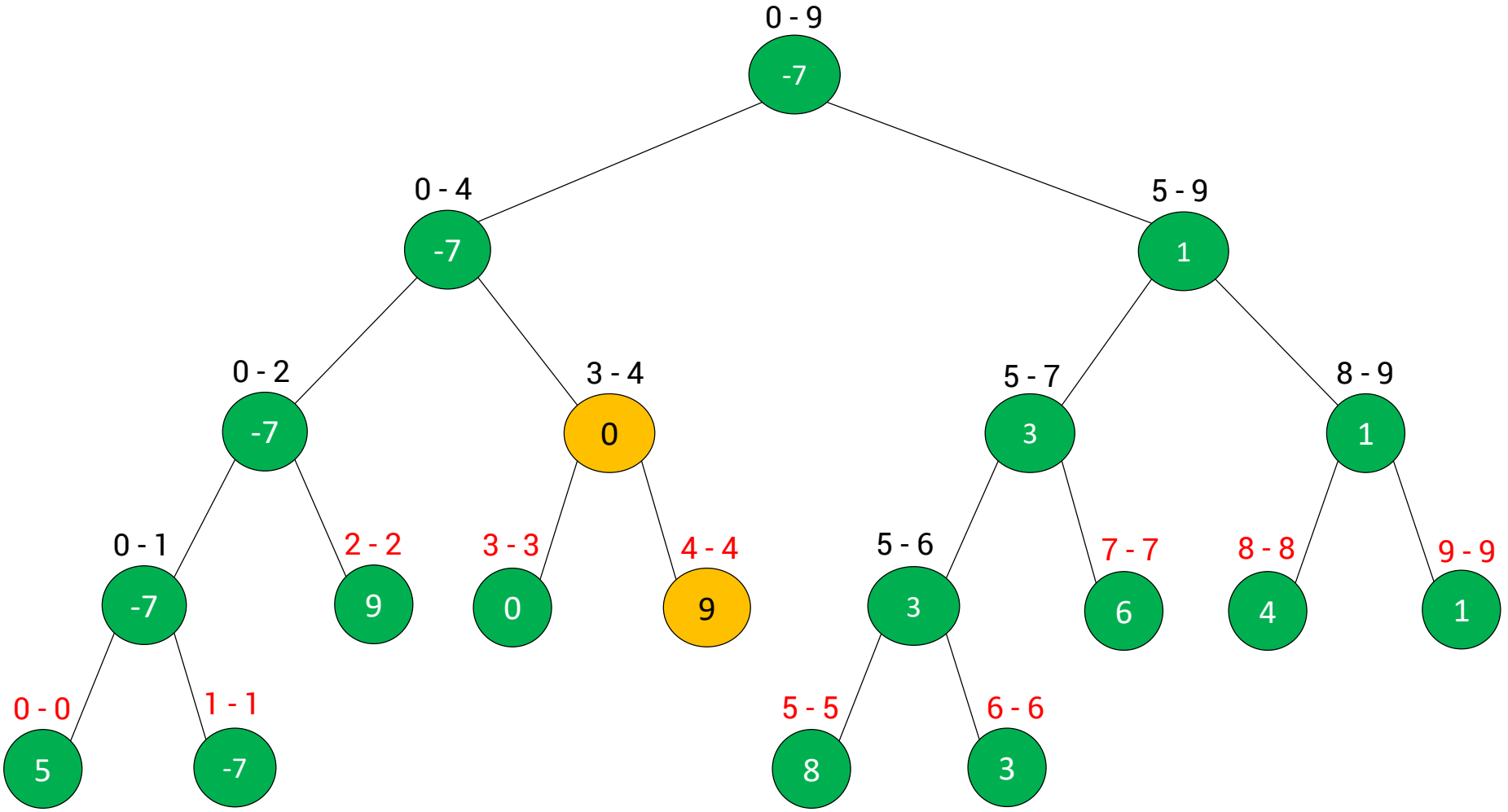
Bước 8: Không cập nhật giá trị [0, 9]

Không thay đổi giá trị [0 – 9] trên Segment ➔ **Dừng thuật toán.**



Kết quả bài toán





Source Code Update RMQ

```
1.  void updateQuery(vector<int> &segtree, vector<int> &a, int left, int right,
                                    int index, int pos, int value) {
2.      if (pos < left || right < pos) {
3.          return;
4.      }
5.      if (left == right) {
6.          a[pos] = value;
7.          segtree[index] = value;
8.          return;
9.      }
10.     int mid = (left + right) / 2;
11.     if (pos <= mid) {
12.         updateQuery(segtree, a, left, mid, 2 * index + 1, pos, value);
13.     }
14.     else //if pos > mid
15.         updateQuery(segtree, a, mid + 1, right, 2 * index + 2, pos, value);
16.
17.     segtree[index] = min(segtree[2 * index + 1], segtree[2 * index + 2]);
18. }
```

Source Code Update RMQ

```
17. int minRange(vector<int>& segtree, int left, int right, int from, int to, int index) {
18.     if (from <= left && to >= right) {
19.         return segtree[index];
20.     }
21.     if (from > right || to < left) {
22.         return INF;
23.     }
24.     int mid = (left + right) / 2;
25.     return min(minRange(segtree, mid + 1, right, from, to, 2 * index + 2),
26.         minRange(segtree, left, mid, from, to, 2 * index + 1));
27. }
```

Source Code Update RMQ

```
28. void buildTree(vector<int>& a, vector<int> &segtree, int left, int right, int index) {
29.     if (left == right) {
30.         segtree[index] = a[left];
31.         return;
32.     }
33.     int mid = (left + right) / 2;
34.     buildTree(a, segtree, left, mid, 2 * index + 1);
35.     buildTree(a, segtree, mid + 1, right, 2 * index + 2);
36.     segtree[index] = min(segtree[2 * index + 1], segtree[2 * index + 2]);
37. }
```

Source Code Update RMQ

```
38. int main() {
39.     vector<int> a = { 5, -7, 9, 0, -2, 8, 3, 6, 4, 1 };
40.     int n = a.size();
41.     //Height of segment tree
42.     int h = (int)ceil(log2(n));
43.     //Maximum size of segment tree
44.     int sizeTree = 2 * (int)pow(2, h) - 1;
45.     vector<int> segtree(sizeTree, INF);
46.     buildTree(a, segtree, 0, n - 1, 0);
47.     int fromRange = 2;
48.     int toRange = 7;
49.     int min = minRange(segtree, 0, n - 1, fromRange, toRange, 0);
50.     cout << "Before update" << endl;
51.     cout << "Range minimum query: " << min << endl;
```

Source Code Update RMQ

```
52.       //Position update
53.       int pos = 4;
54.       //Value update
55.       int value = 9;
56.       updateQuery(segtree, a, 0, n - 1, 0, pos, value);
57.       min = minRange(segtree, 0, n - 1, fromRange, toRange, 0);
58.       cout << "After update" << endl;
59.       cout << "Range minimum query: " << min << endl;
60.       return 0;
61.   }
```

Source Code Update RMQ

```
1.  from math import ceil, log2
2.  INF = 10**9
3.  def updateQuery(segtree, a, left, right, index, pos, value):
4.      if pos < left or right < pos:
5.          return
6.
7.      if left == right:
8.          a[pos] = value
9.          segtree[index] = value
10.         return
11.
12.     mid = (left + right) // 2
13.     if pos <= mid:
14.         updateQuery(segtree, a, left, mid, 2 * index + 1, pos, value)
15.     else:  # if pos > mid
16.         updateQuery(segtree, a, mid + 1, right, 2 * index + 2, pos, value)
17.
18.     segtree[index] = min(segtree[2 * index + 1], segtree[2 * index + 2])
```

Source Code Update RMQ

```
19. def minRange(segtree, left, right, fr, to, index):
20.     if fr <= left and right <= to:
21.         return segtree[index]
22.     if fr > right or to < left:
23.         return INF
24.     mid = (left + right) // 2
25.     return min(minRange(segtree, left, mid, fr, to, 2 * index + 1),
26.                minRange(segtree, mid + 1, right, fr, to, 2 * index + 2))
27.
28.  def buildTree(a, segtree, left, right, index):
29.     if left == right:
30.         segtree[index] = a[left]
31.         return
32.     mid = (left + right) // 2
33.     buildTree(a, segtree, left, mid, 2 * index + 1)
34.     buildTree(a, segtree, mid + 1, right, 2 * index + 2)
35.     segtree[index] = min(segtree[2 * index + 1], segtree[2 * index + 2])
```

Source Code Update RMQ

```
36. if __name__ == '__main__':
37.     a = [5, -7, 9, 0, -2, 8, 3, 6, 4, 1]
38.     n = len(a)
39.     h = ceil(log2(n))
40.     sizeTree = 2 * (2**h) - 1
41.     segtree = [INF] * sizeTree
42.     buildTree(a, segtree, 0, n - 1, 0)
43.     fromRange = 2
44.     toRange = 7
45.     minValue = minRange(segtree, 0, n - 1, fromRange, toRange, 0)
46.     print("Before update")
47.     print("Range minimum query:", minValue)
48.     # Position update
49.     pos = 4
50.     # Value update
51.     value = 9
52.     updateQuery(segtree, a, 0, n - 1, 0, pos, value)
53.     minValue = minRange(segtree, 0, n - 1, fromRange, toRange, 0)
54.     print("After update")
55.     print("Range minimum query:", minValue)
```

Source Code Update RMQ

```java
import java.util.Arrays;
public class Main {
    private static final int INF = (int)1e9;
    private static double log2(int number) {
        return Math.log(number) / Math.log(2);
    }
    private static void updateQuery(int[] segtree, int[] a, int left, int right,
                                    int index, int pos, int value) {
        if (pos < left || right < pos) {
            return;
        }
        if (left == right) {
            a[pos] = value;
            segtree[index] = value;
            return;
        }
        int mid = (left + right) / 2;
        if (pos <= mid) {
            updateQuery(segtree, a, left, mid, 2 * index + 1, pos, value);
        }
        else //if pos > mid
            updateQuery(segtree, a, mid + 1, right, 2 * index + 2, pos, value);
        segtree[index] = Math.min(segtree[2 * index + 1], segtree[2 * index + 2]);
    }
```

Source Code Update RMQ

```
24.     private static int minRange(int[] segtree, int left, int right, int f
                                    int to, int index) {
25.         if (from <= left && to >= right) {
26.             return segtree[index];
27.         }
28.         if (from > right || to < left) {
29.             return INF;
30.         }
31.         int mid = (left + right) / 2;
32.         int a = minRange(segtree, left, mid, from, to, 2 * index + 1);
33.         int b = minRange(segtree, mid + 1, right, from, to, 2 * index + 2);
34.         return Math.min(a, b);
35.     }
```

Source Code Update RMQ

```
36.     private static void buildTree(int[] a, int[] segtree, int left, int right,
                                      int index) {
37.         if (left == right) {
38.             segtree[index] = a[left];
39.             return;
40.         }
41.         int mid = (left + right) / 2;
42.         buildTree(a, segtree, left, mid, 2 * index + 1);
43.         buildTree(a, segtree, mid + 1, right, 2 * index + 2);
44.         segtree[index] = Math.min(segtree[2 * index + 1], segtree[2 * index + 2]);
45.     }
```

Source Code Update RMQ

```
46.     public static void main(String[] args) {
47.         int[] a = new int[]{ 5, -7, 9, 0, -2, 8, 3, 6, 4, 1 };
48.         int n = a.length;
49.         //Height of segment tree
50.         int h = (int)Math.ceil(log2(n));
51.         //Maximum size of segment tree
52.         int sizeTree = 2 * (int)Math.pow(2, h) - 1;
53.         int[] segtree = new int[sizeTree];
54.         Arrays.fill(segtree, INF);
55.         buildTree(a, segtree, 0, n - 1, 0);
56.         int fromRange = 2;
57.         int toRange = 7;
58.         int min = minRange(segtree, 0, n - 1, fromRange, toRange, 0);
59.         System.out.println("Before update");
60.         System.out.printf("Range minimum query: %d\n", min);
```

Source Code Update RMQ

```
61.             //Position update
62.             int pos = 4;
63.             //Value update
64.             int value = 9;
65.             updateQuery(segtree, a, 0, n - 1, 0, pos, value);
66.
67.             min = minRange(segtree, 0, n - 1, fromRange, toRange, 0);
68.             System.out.println("After update");
69.             System.out.printf("Range minimum query: %d\n", min);
70.         }
```

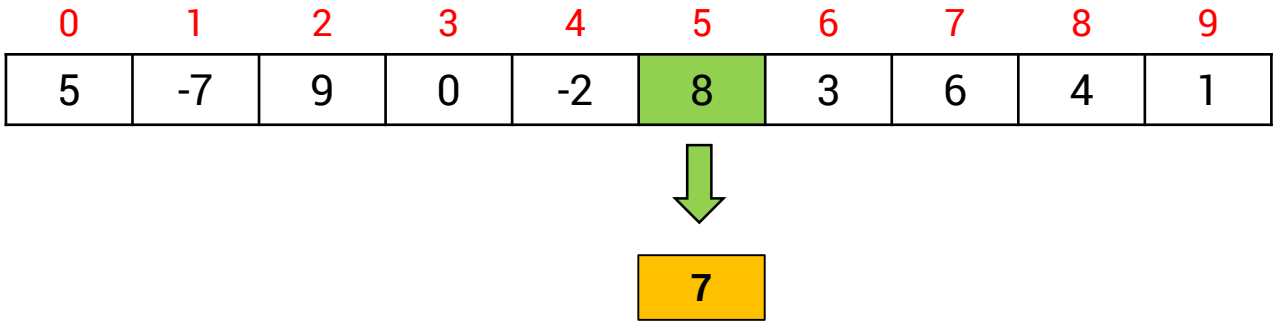
2. UPDATE

SUM OF GIVEN RANGE

Update: Sum of given range

Big-O Orange

Update SGR: Thao tác này dùng để cập nhật một giá trị bất kỳ trên Segment Tree, sau đó sử dụng lại truy vấn SGR kết quả sẽ được cập nhật theo giá trị mới nhất trên cây.

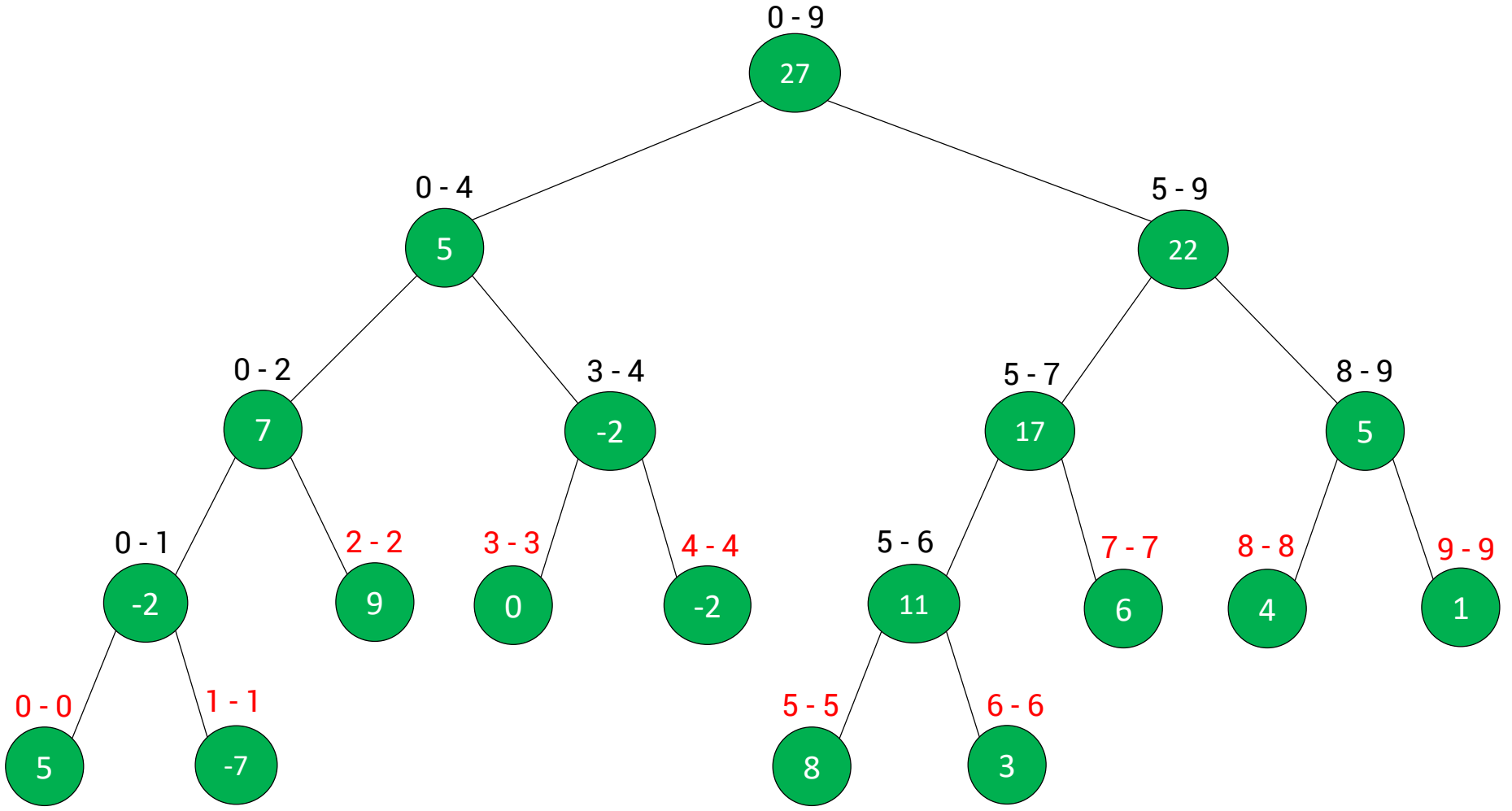


- Truy vấn [3, 8]: 19
- Cập nhật giá trị tại vị trí 5: 8→ 7
- Truy vấn [3, 8]: 18

Time complexity: O(LogN)

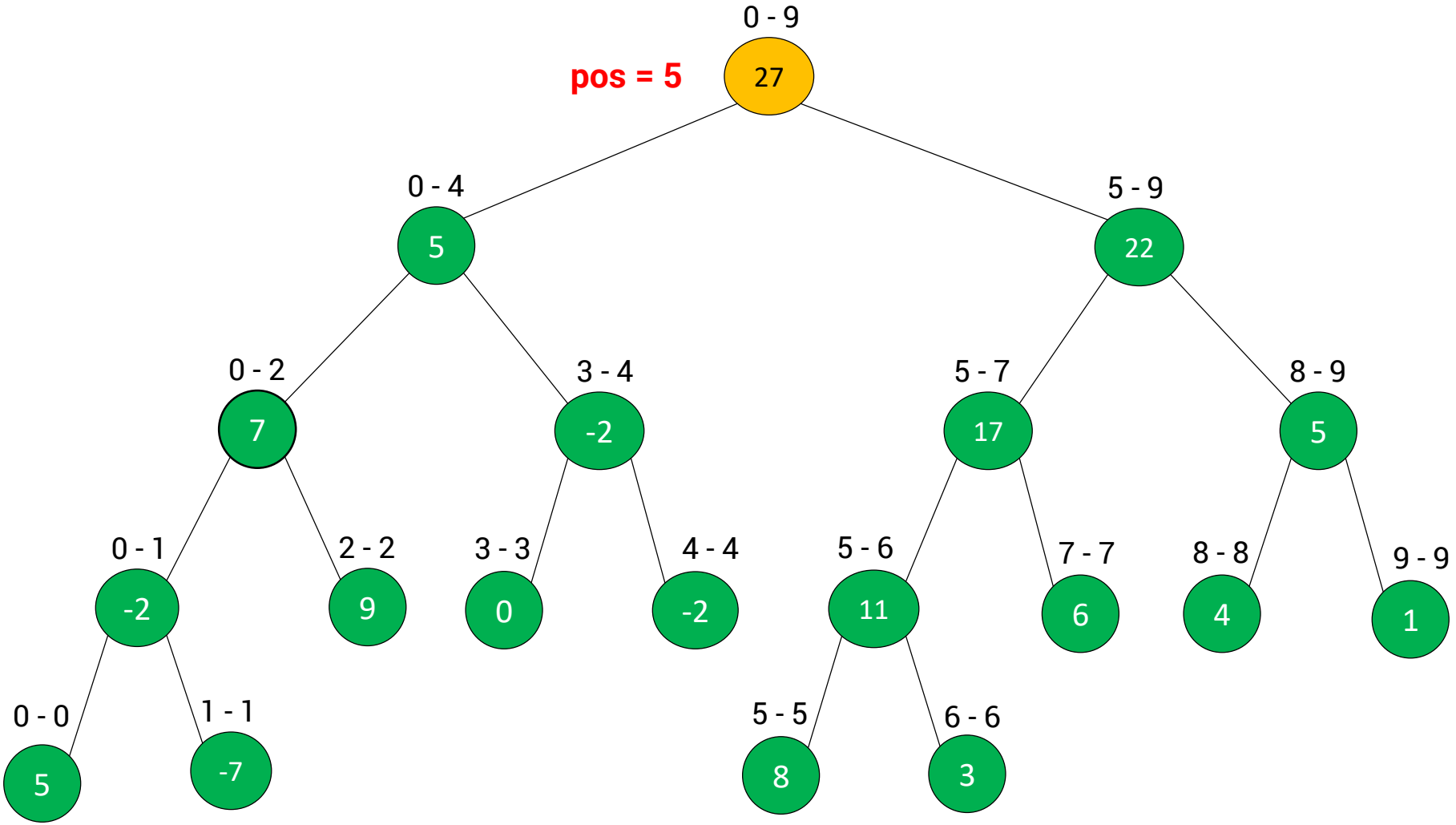
Bước 0: Xây dựng Segment Tree

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1



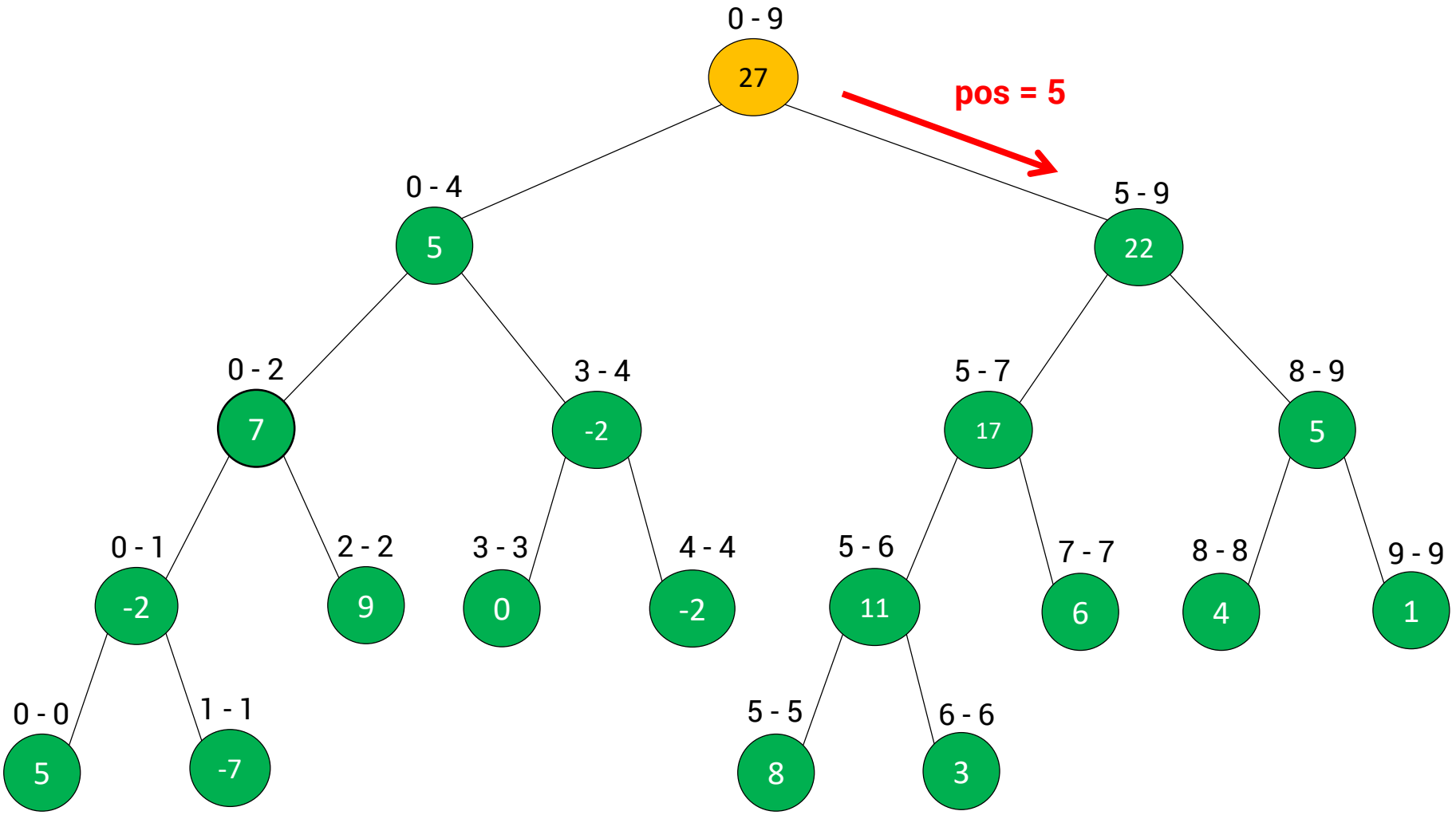
Bước 1: Bắt đầu chạy từ Node gốc của Segment Tree

Từ vị trí node gốc xét xem **position** cần cập nhật sẽ thuộc về nhánh nào.



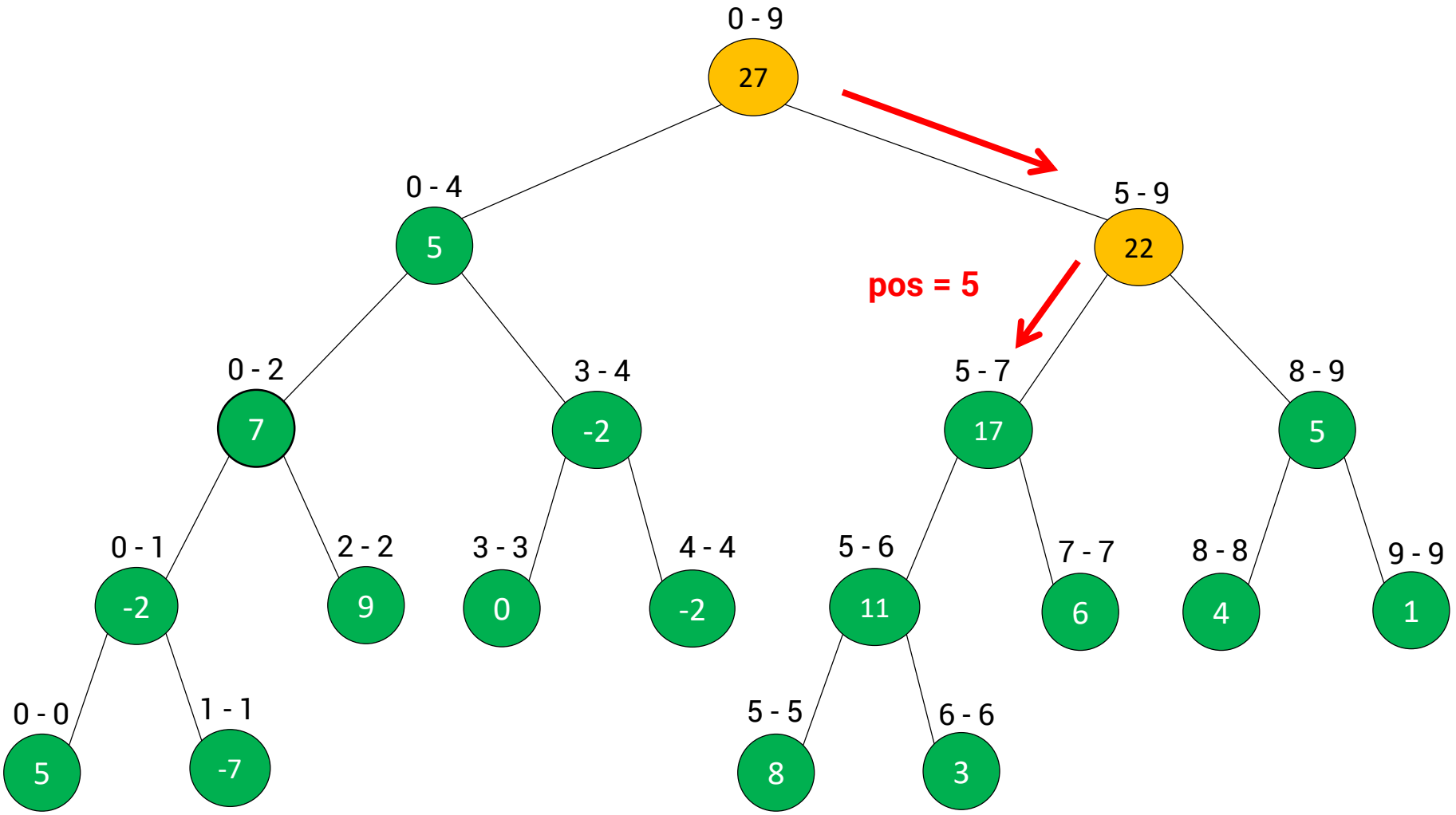
Bước 2: Đi về hướng phải

mid = (left + right) / 2 = (0 + 9) / 2 = 4 < pos(5) ➔ Đi về hướng phải.



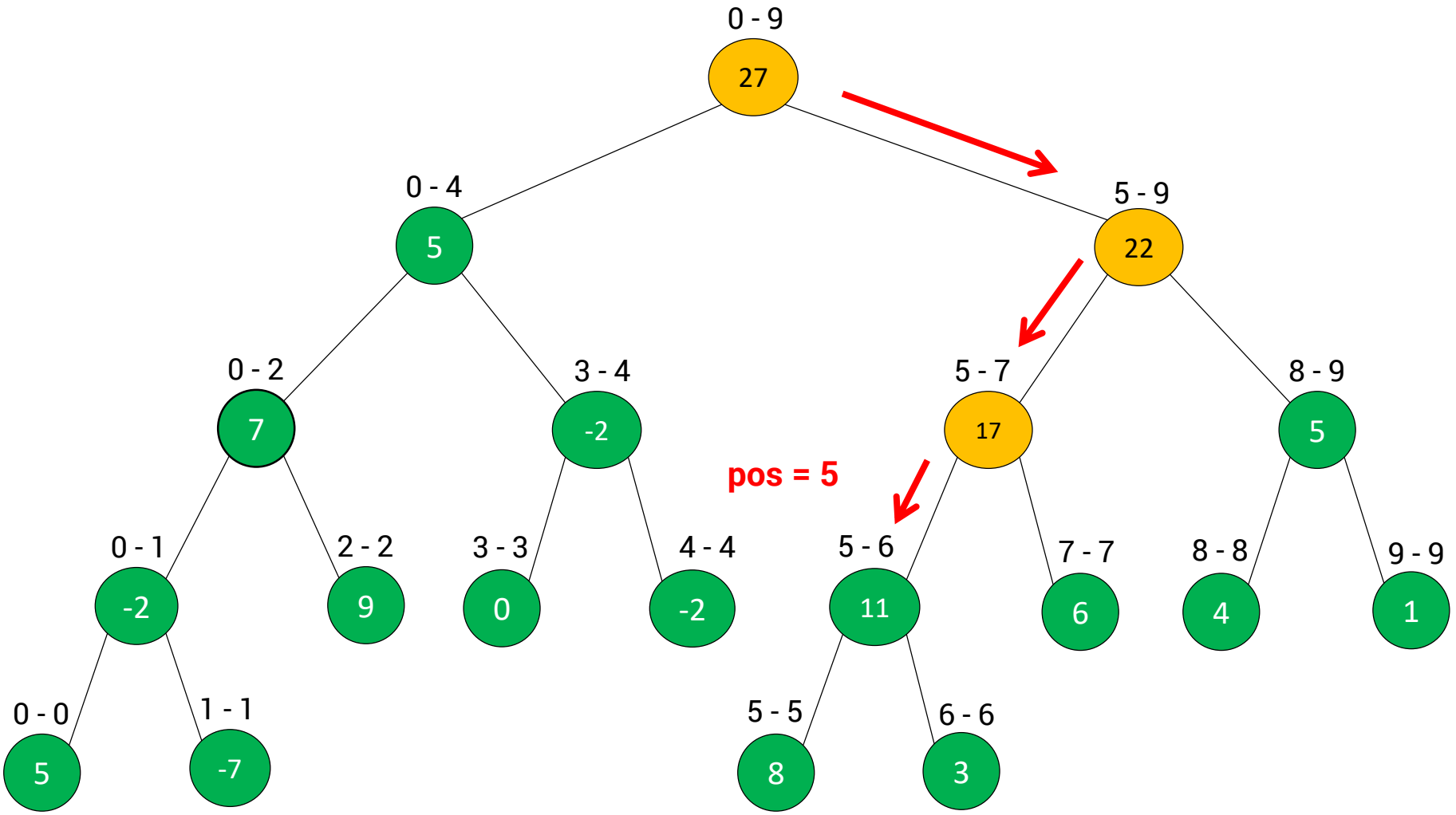
Bước 3: Đi về hướng trái

mid = (left + right) / 2 = (5 + 9) / 2 = 7 ≥ pos(5) ➔ Đi về hướng trái.



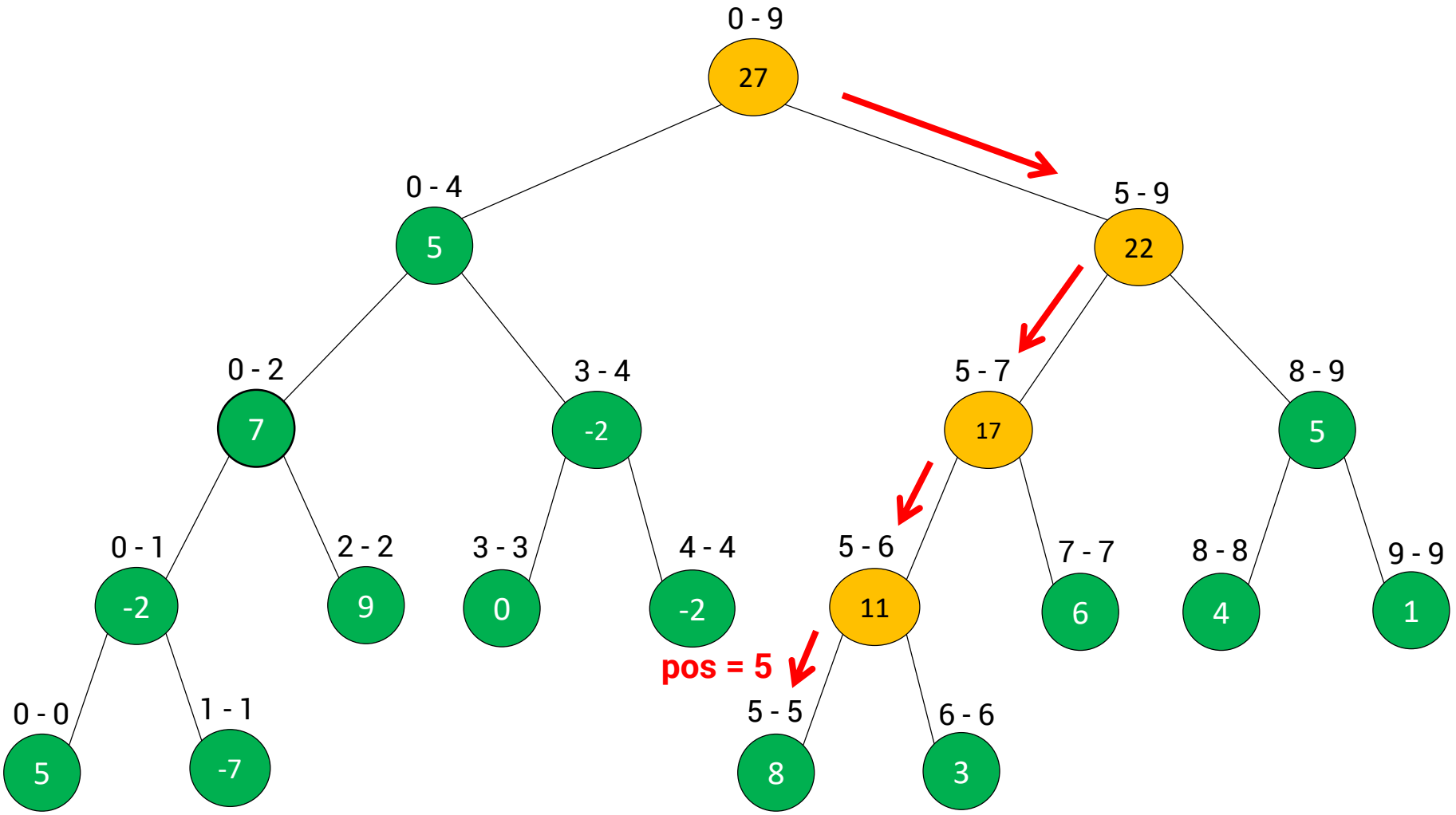
Bước 4: Đi về hướng trái

mid = (left + right) / 2 = (5 + 7) / 2 = 6 ≥ pos(5) ➔ Đi về hướng trái.



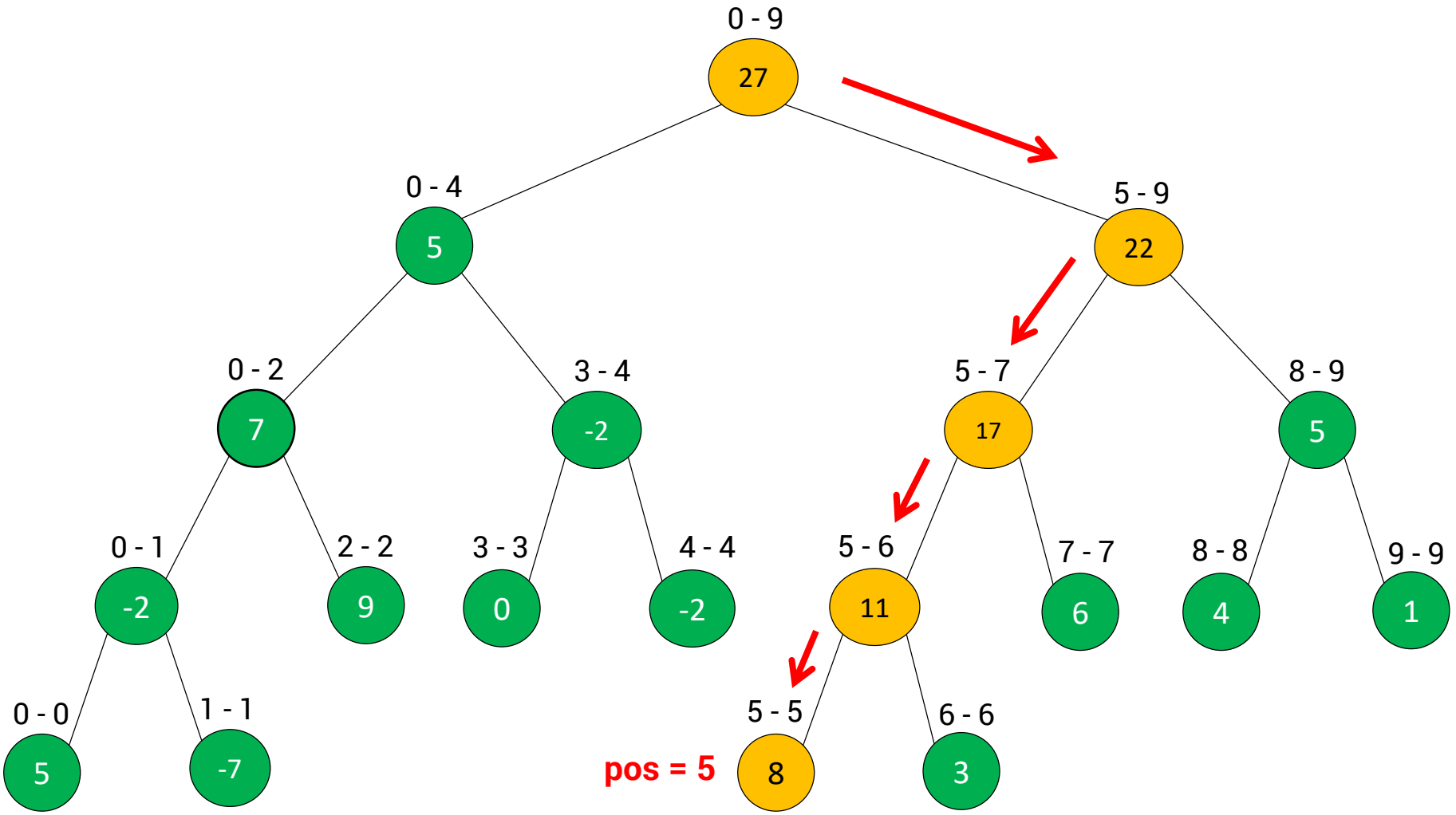
Bước 5: Đi về hướng trái

mid = (left + right) / 2 = (5 + 6) / 2 = 5 ≥ pos(5) ➔ Đi về hướng trái.



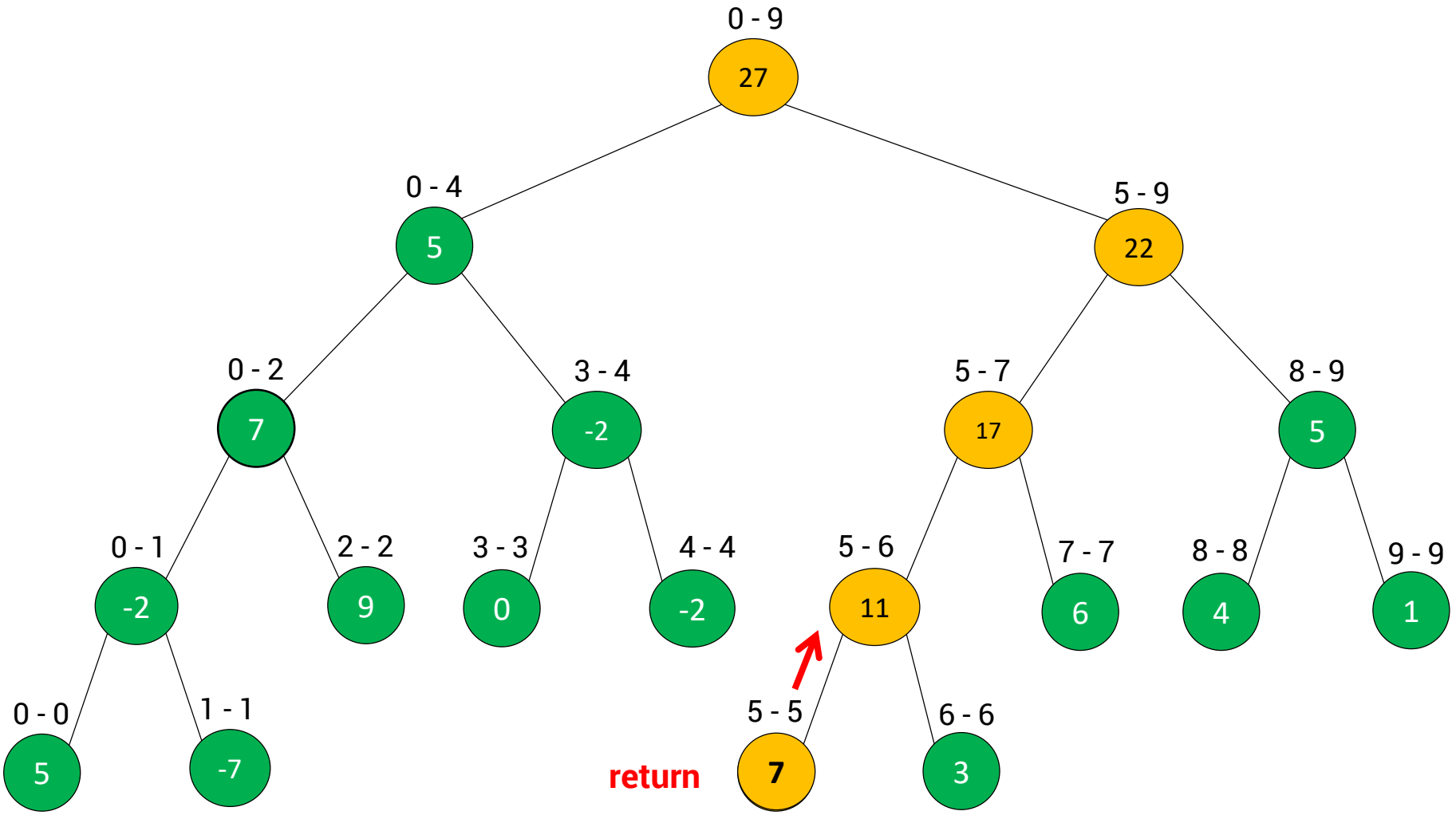
Bước 6: Đến được vị trí pos trên Segment

Tìm được vị trí pos(5) trên Segment Tree.



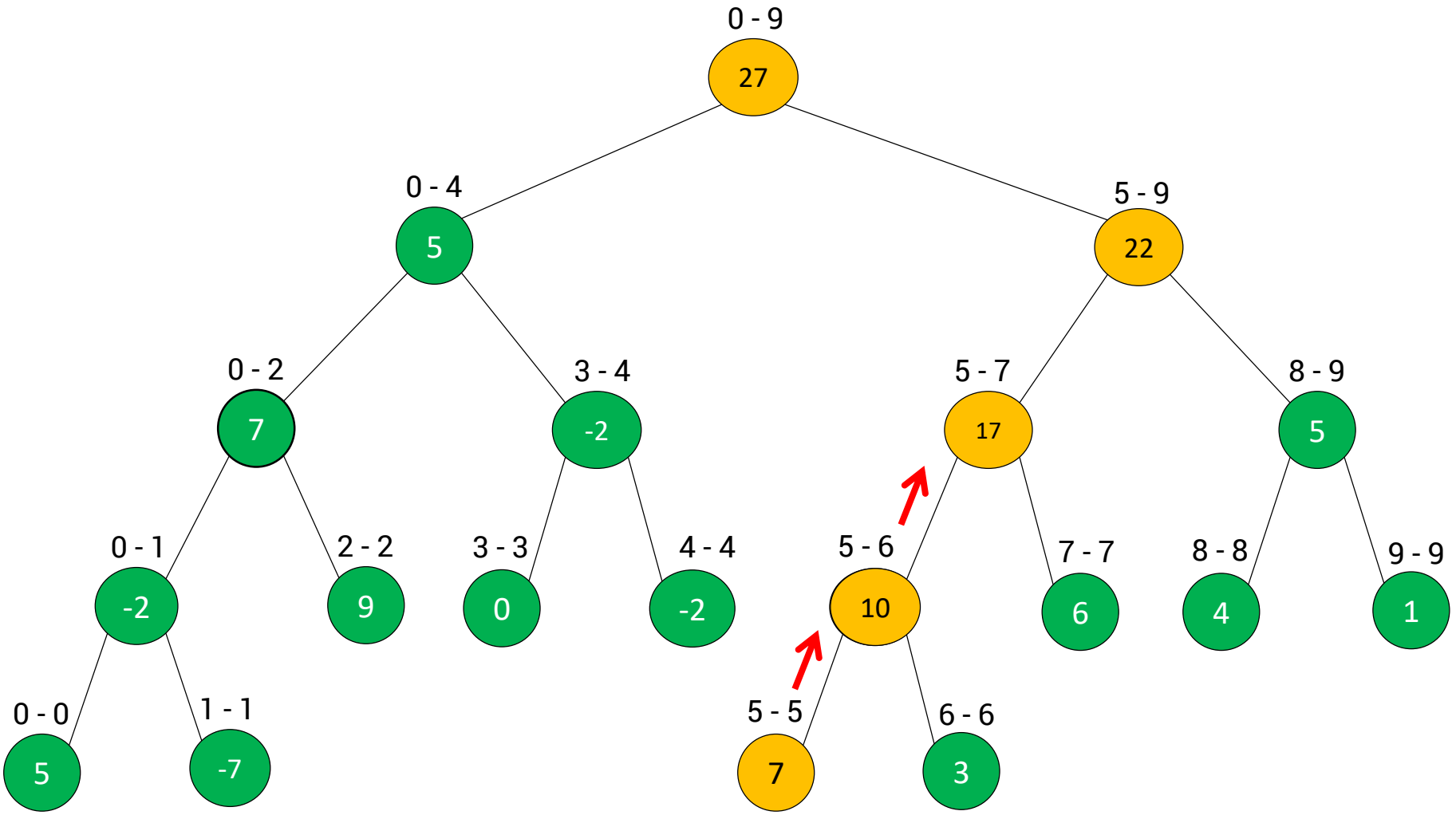
Bước 7: Cập nhật đoạn [5 – 5]

Thay đổi giá trị [5 – 5] trên Segment Tree = 7.



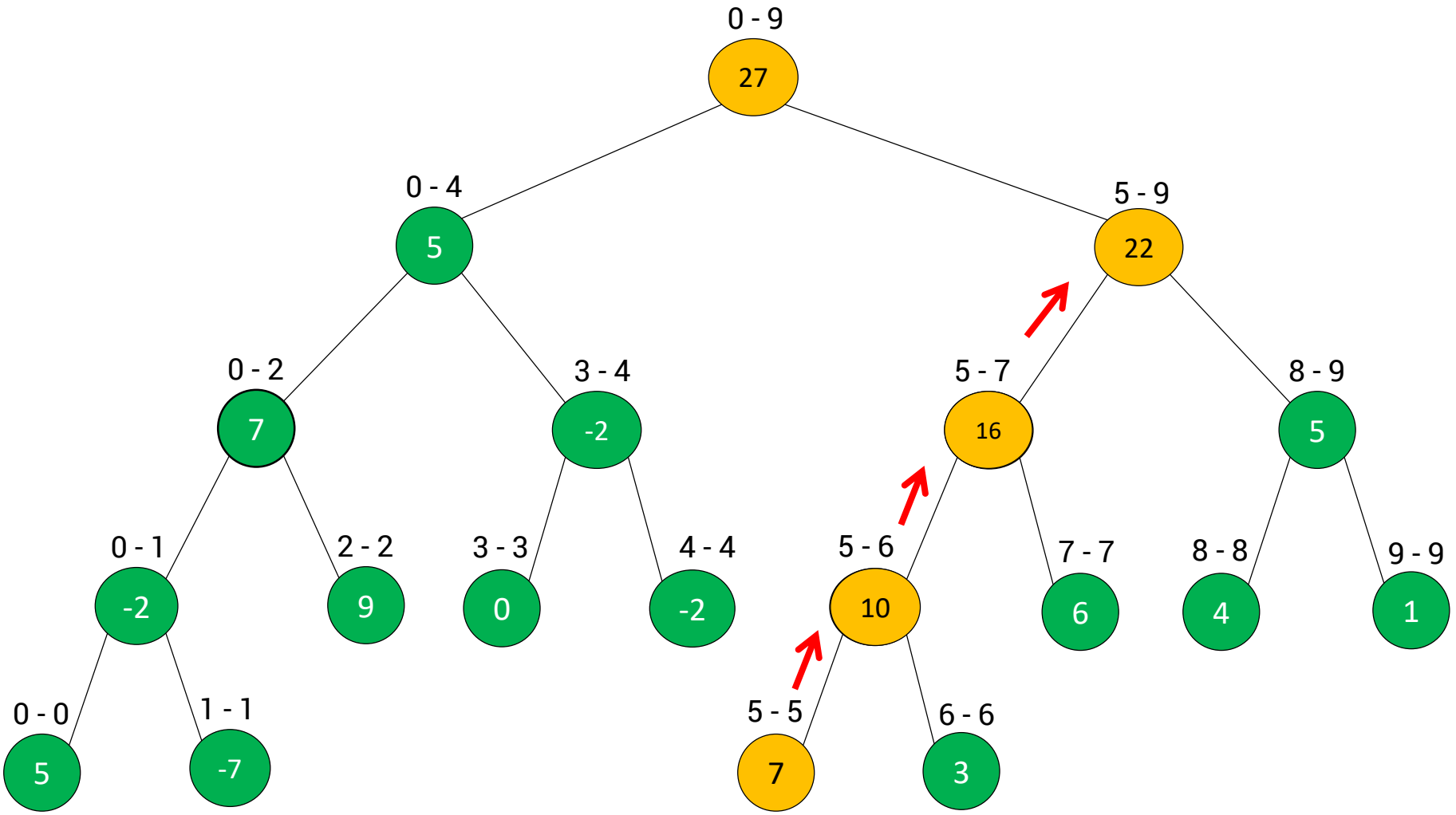
Bước 8: Cập nhật đoạn [5 – 6]

Thay đổi giá trị [5 – 6] trên Segment = 10.



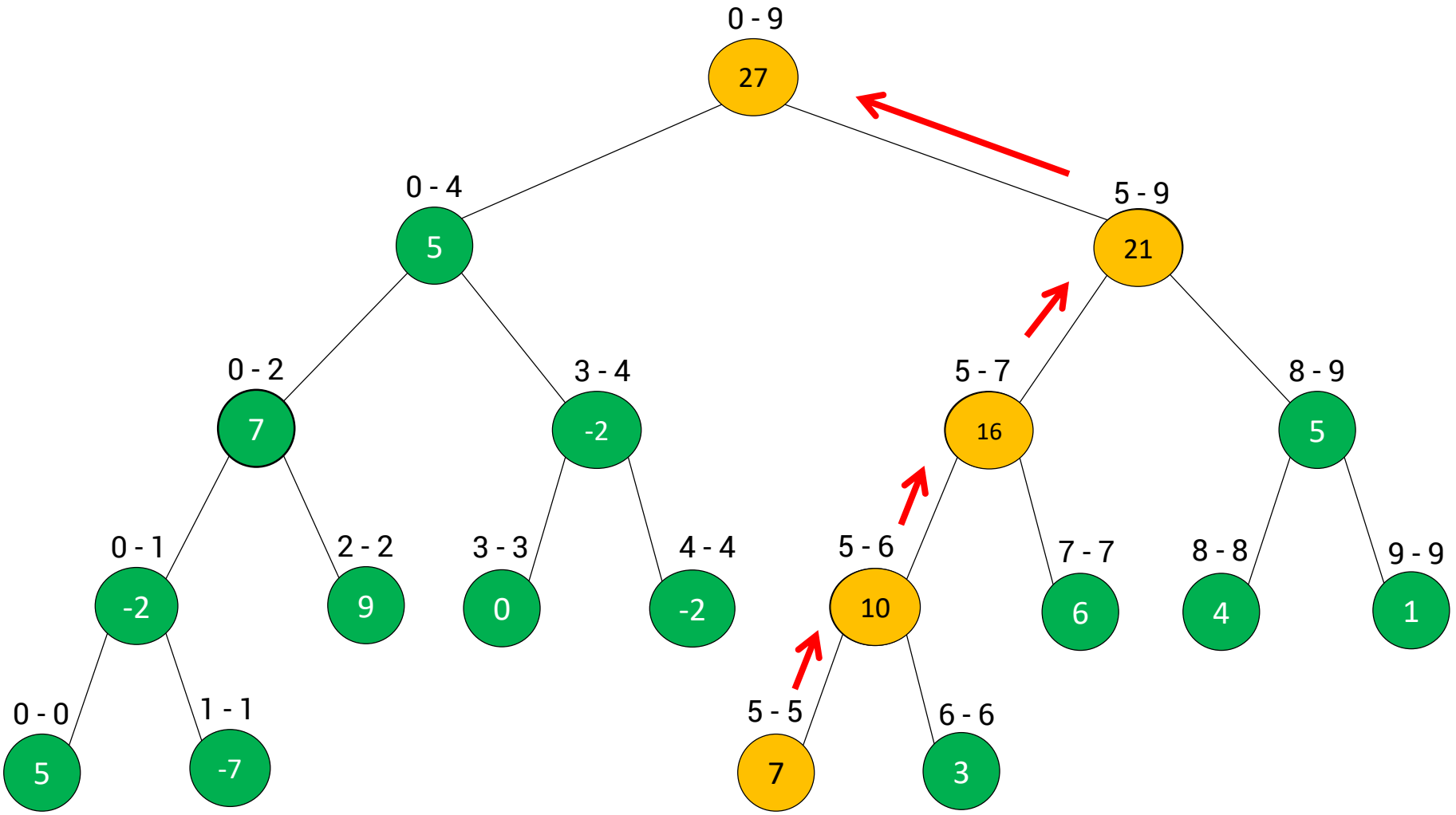
Bước 9: Cập nhật đoạn [5 – 7]

Thay đổi giá trị [5 – 7] trên Segment = 16.



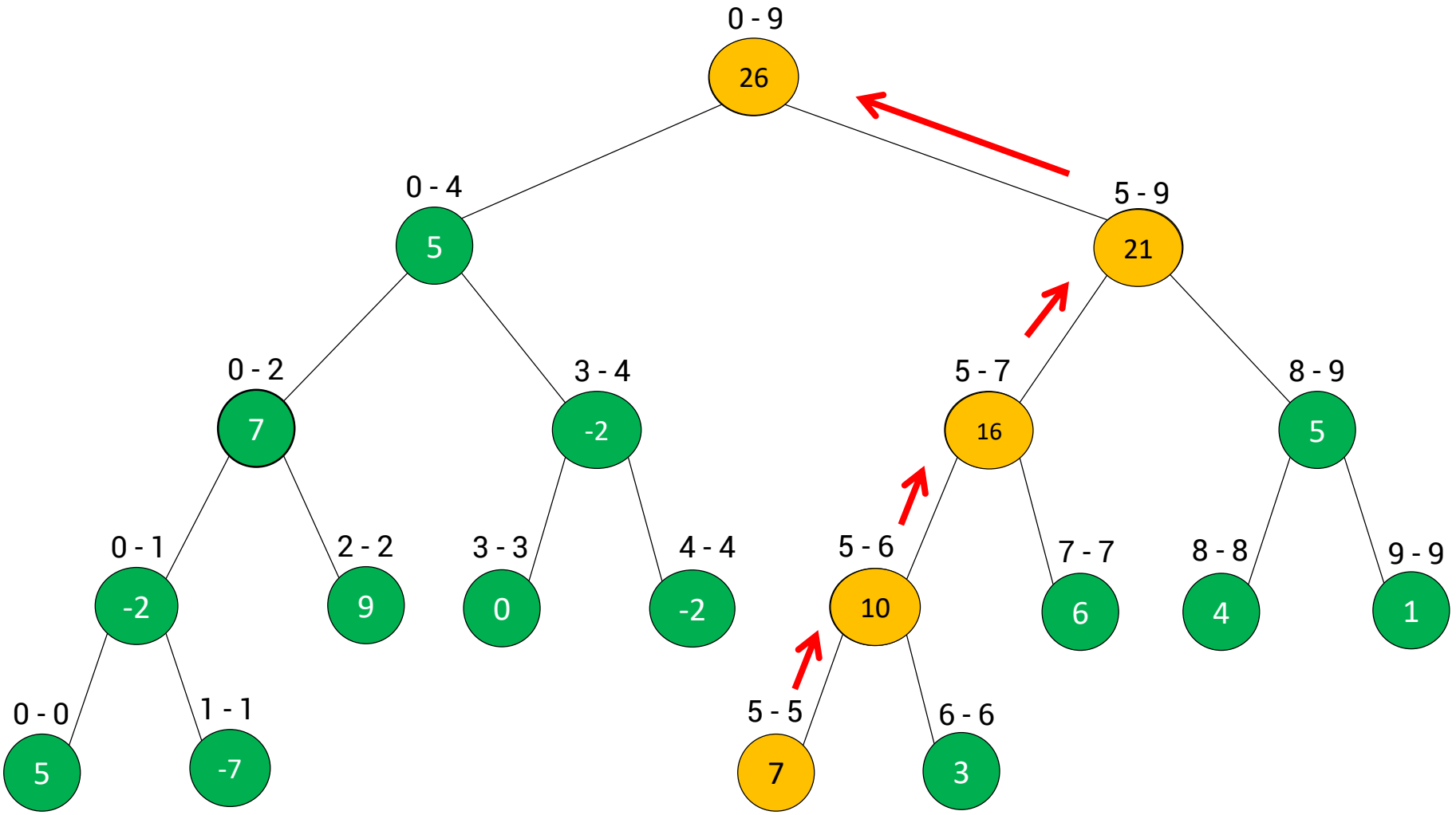
Bước 10: Cập nhật đoạn [5 – 9]

Thay đổi giá trị [5 – 9] trên Segment = 21.



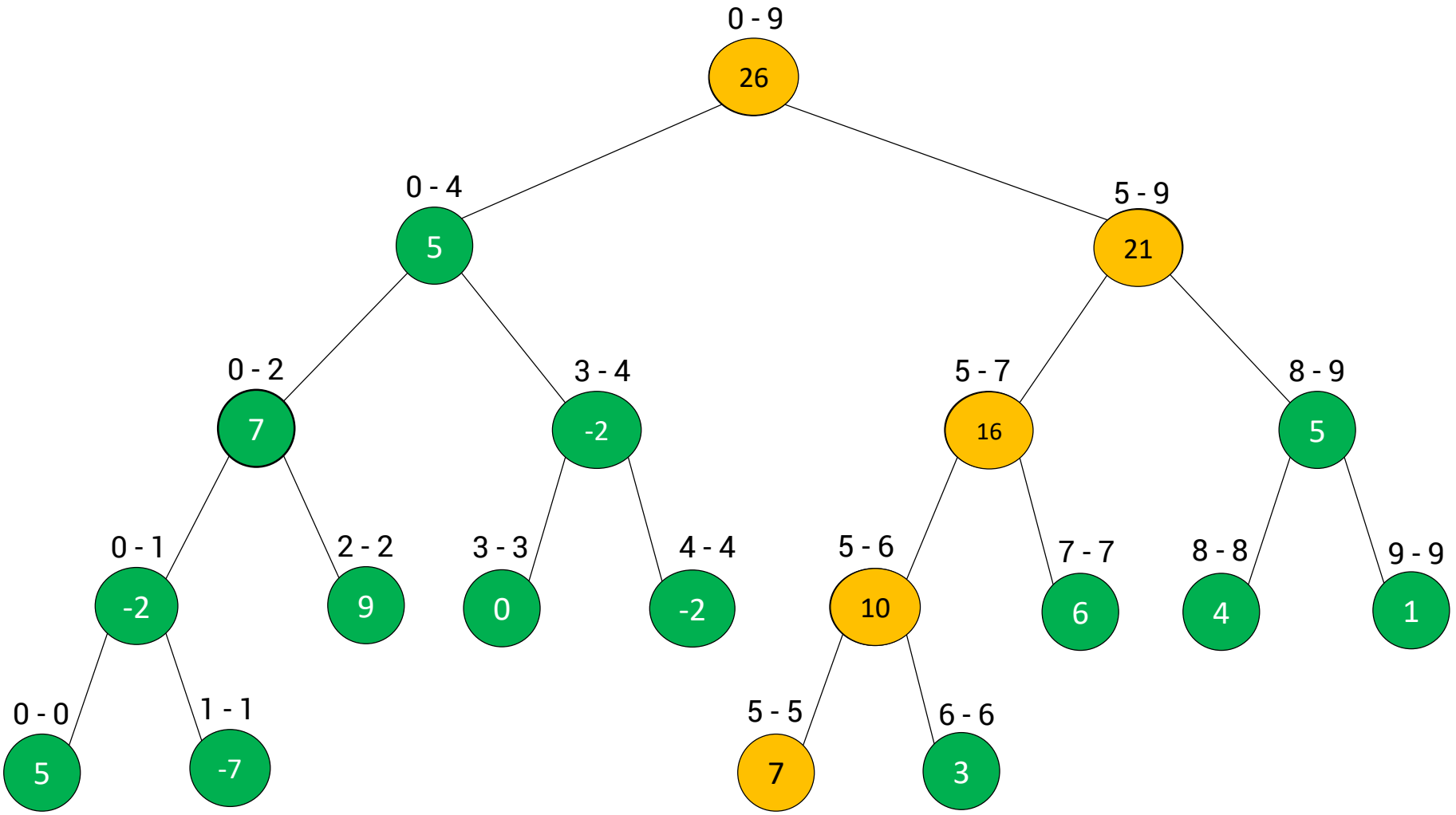
Bước 11: Cập nhật lại node gốc [0 – 9]

Thay đổi giá trị [0 – 9] trên Segment Tree= 26 ➔ **Dừng thuật toán.**



Kết quả bài toán

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	7	3	6	4	1



Source Code Update SGR

```
1.  void updateQuery(vector<int> &segtree, vector<int> &a, int left, int right,
                     int index, int pos, int value) {
2.      if (pos < left || right < pos) {
3.          return;
4.      }
5.      if (left == right) {
6.          a[pos] = value;
7.          segtree[index] = value;
8.          return;
9.      }
10.     int mid = (left + right) / 2;
11.     if (pos <= mid) {
12.         updateQuery(segtree, a, left, mid, 2 * index + 1, pos, value);
13.     }
14.     else //if pos > mid
15.         updateQuery(segtree, a, mid + 1, right, 2 * index + 2, pos, value);
16.
17.     segtree[index] = segtree[2 * index + 1] + segtree[2 * index + 2];
18. }
```

Source Code Update SGR

```
19. int sumRange(vector<int> &segtree, int left, int right, int from, int to, int index) {
20.     if (from <= left && to >= right) {
21.         return segtree[index];
22.     }
23.     if (from > right || to < left) {
24.         return 0;
25.     }
26.     int mid = (left + right) / 2;
27.     return sumRange(segtree, left, mid, from, to, 2 * index + 1)
                    + sumRange(segtree, mid + 1, right, from, to, 2 * index + 2);
28. }
```

Source Code Update SGR

```
1.  def updateQuery(segtree, a, left, right, index, pos, value):
2.      if pos < left or right < pos:
3.          return
4.      if left == right:
5.          a[pos] = value
6.          segtree[index] = value
7.          return
8.      mid = (left + right) // 2
9.      if pos <= mid:
10.         updateQuery(segtree, a, left, mid, 2 * index + 1, pos, value)
11.     else: # if pos > mid
12.         updateQuery(segtree, a, mid + 1, right, 2 * index + 2, pos, value)
13.     segtree[index] = segtree[2 * index + 1] + segtree[2 * index + 2]
```

Source Code Update SGR

```
14. def sumRange(segtree, left, right, fr, to, index):
15.     if fr <= left and to >= right:
16.         return segtree[index]
17.     if fr > right or to < left:
18.         return 0
19.     mid = (left + right) // 2
20.     return sumRange(segtree, left, mid, fr, to, 2 * index + 1)
                    + sumRange(segtree, mid + 1, right, fr, to, 2 * index + 2)
```

Source Code Update SGR

```
1.  private static void updateQuery(int[] segtree, int[] a, int left, int right,
                                   int index, int pos, int value) {
2.          if (pos < left || right < pos) {
3.              return;
4.          }
5.          if (left == right) {
6.              a[pos] = value;
7.              segtree[index] = value;
8.              return;
9.          }
10.         int mid = (left + right) / 2;
11.         if (pos <= mid) {
12.             updateQuery(segtree, a, left, mid, 2 * index + 1, pos, value);
13.         }
14.         else //if pos > mid
15.             updateQuery(segtree, a, mid + 1, right, 2 * index + 2, pos, value);
16.
17.         segtree[index] = segtree[2 * index + 1] + segtree[2 * index + 2];
18.     }
```

Source Code Update SGR

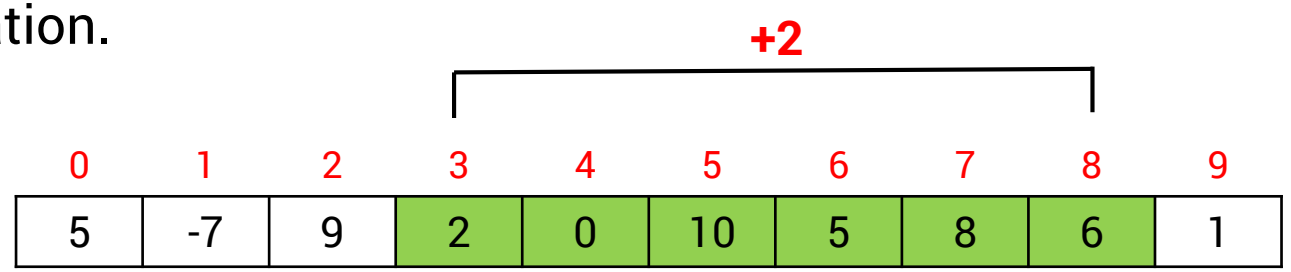
```
19.     private static int sumRange(int[] segtree, int left, int right,
                                    int from, int to, int index) {
20.         if (from <= left && to >= right) {
21.             return segtree[index];
22.         }
23.         if (from > right || to < left) {
24.             return 0;
25.         }
26.         int mid = (left + right) / 2;
27.         return sumRange(segtree, left, mid, from, to, 2 * index + 1)
28.                 + sumRange(segtree, mid + 1, right, from, to, 2 * index + 2);
29.     }
```

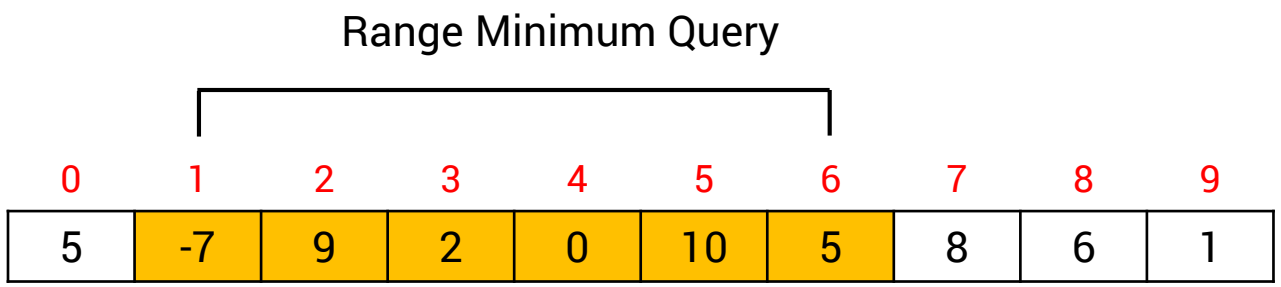
3. LAZY PROPAGATION

RANGE MINIMUM QUERY

Lazy Propagation - Range minimum query

Lazy Propagation - RMQ: Cập nhật đoạn bất kỳ delta đơn vị, sau đó tìm giá trị nhỏ nhất của một đoạn bất kỳ trên dãy số với kỹ thuật Lazy Propagation.





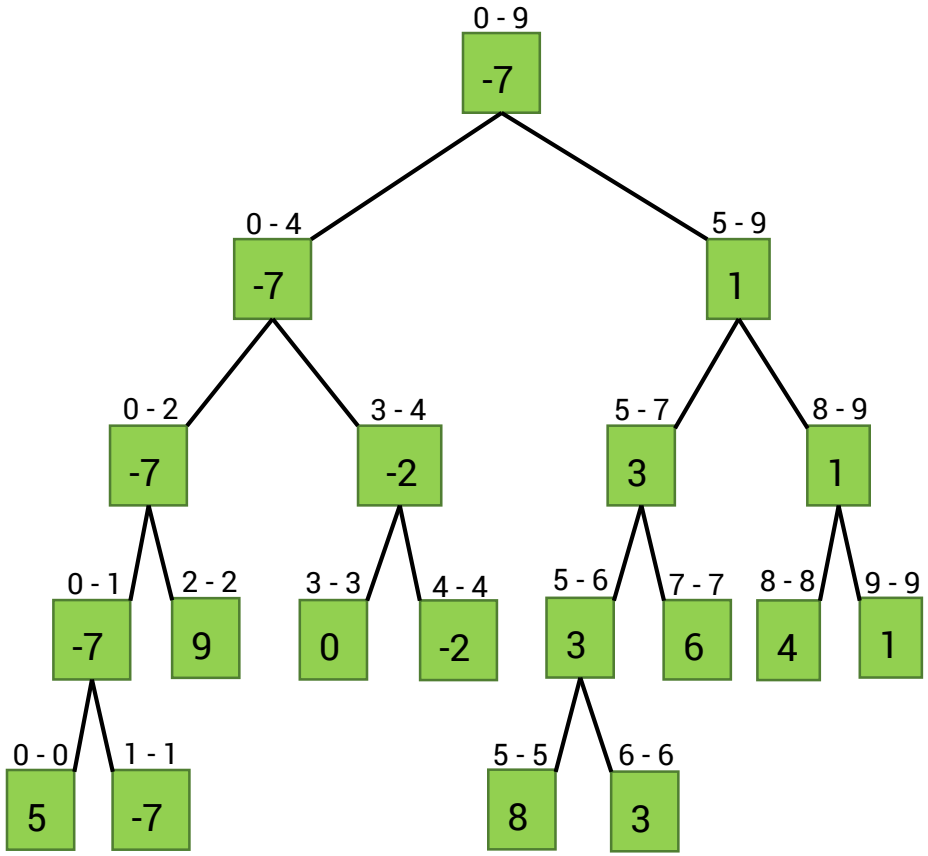
Giá trị nhỏ nhất trong đoạn [1, 6]: -7

Time complexity: O(LogN)

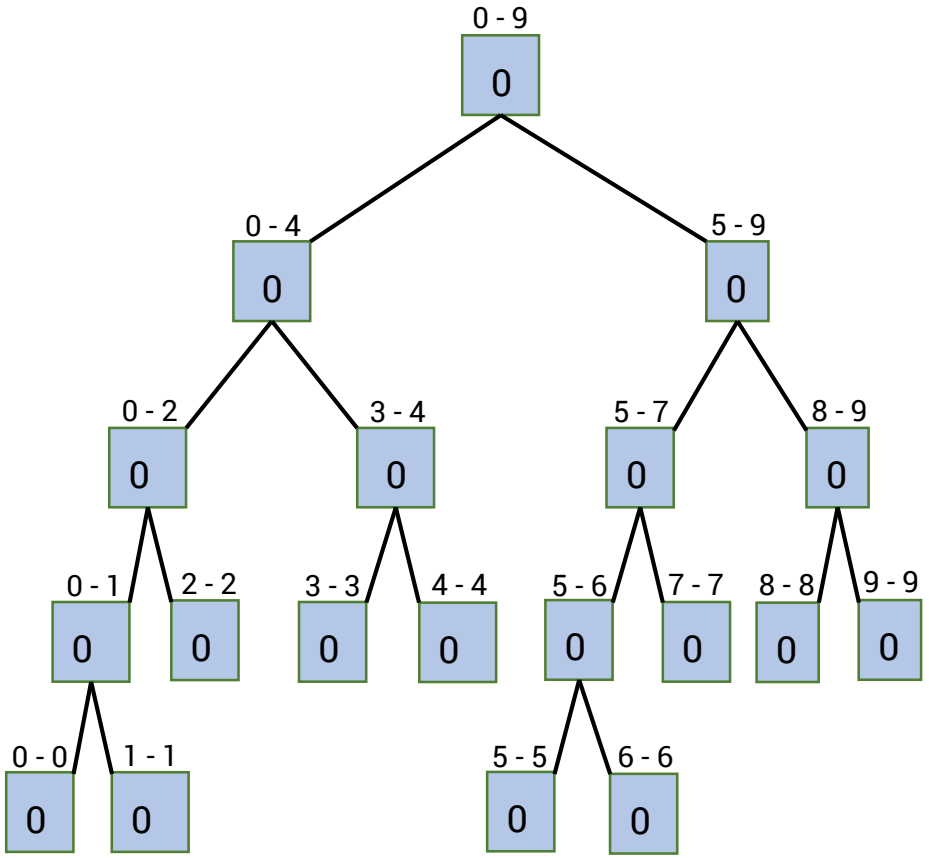
Bước 0: Xây dựng Segment Tree - RQM

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1

Segment Tree

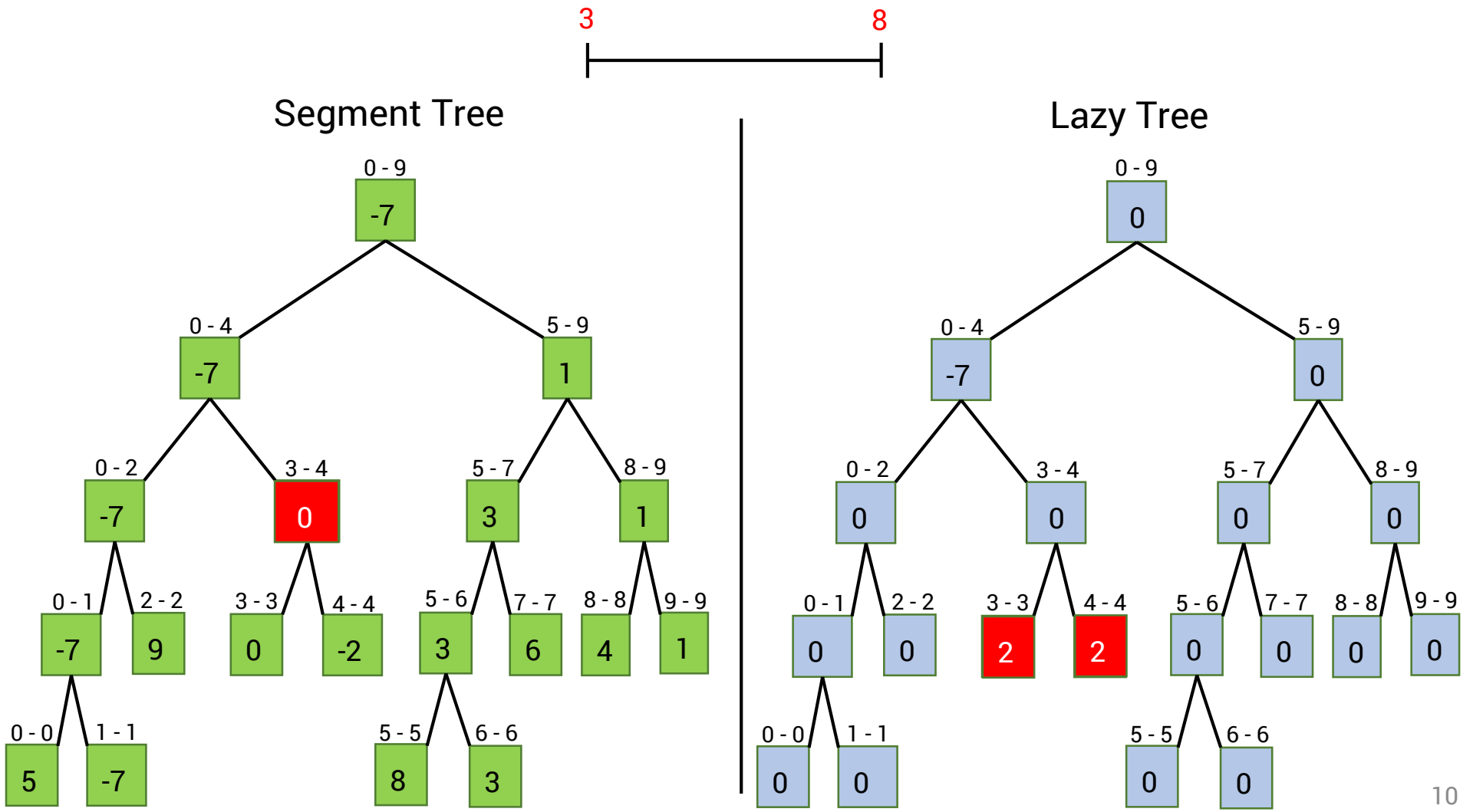


Lazy Tree



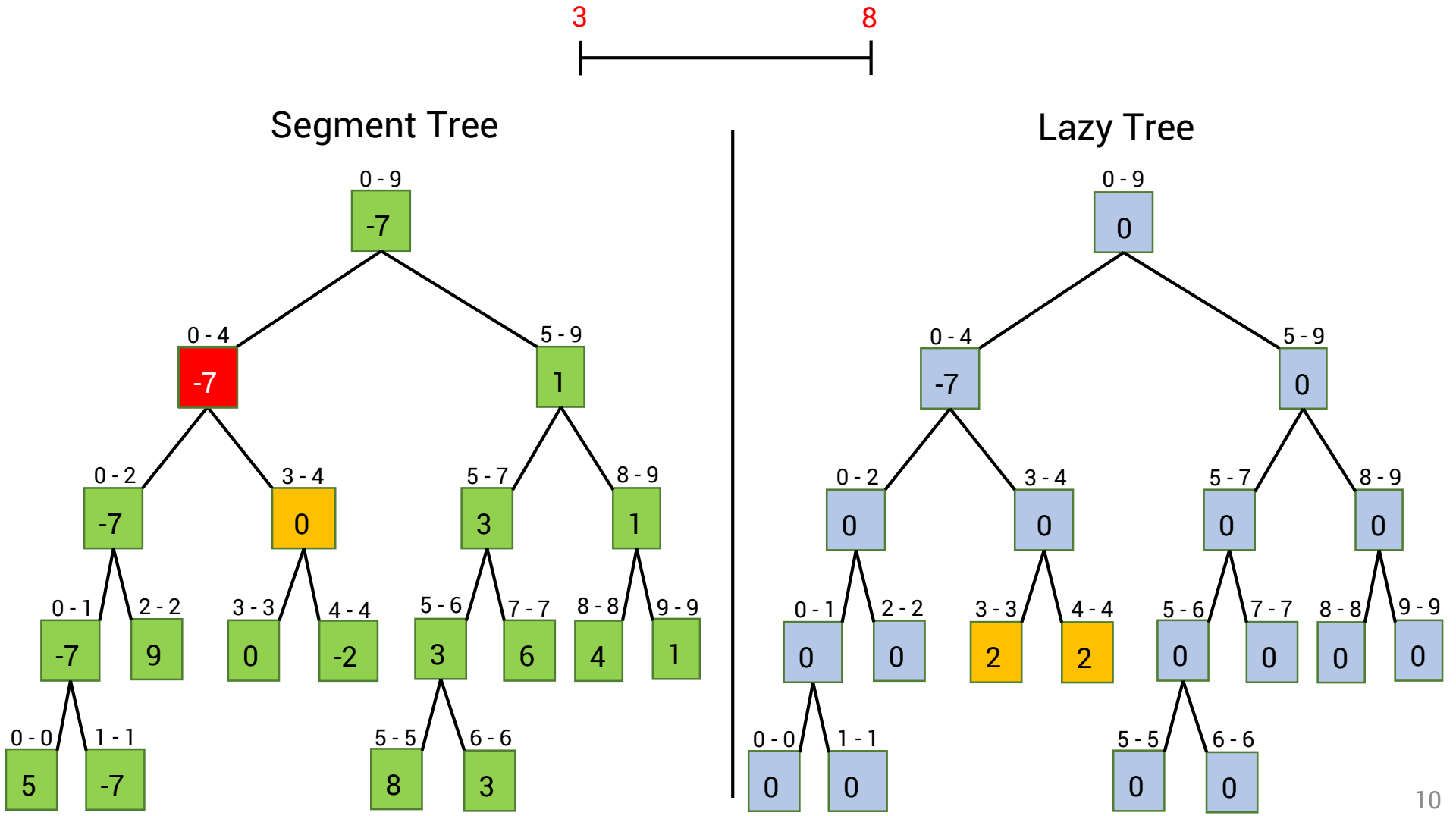
Bước 1: Cập nhật node (3, 4)

Tìm đến node (3, 4) của Segment Tree để tăng giá trị lên (+2). Sau đó tăng giá trị (+2) trên node con của cây Lazy.



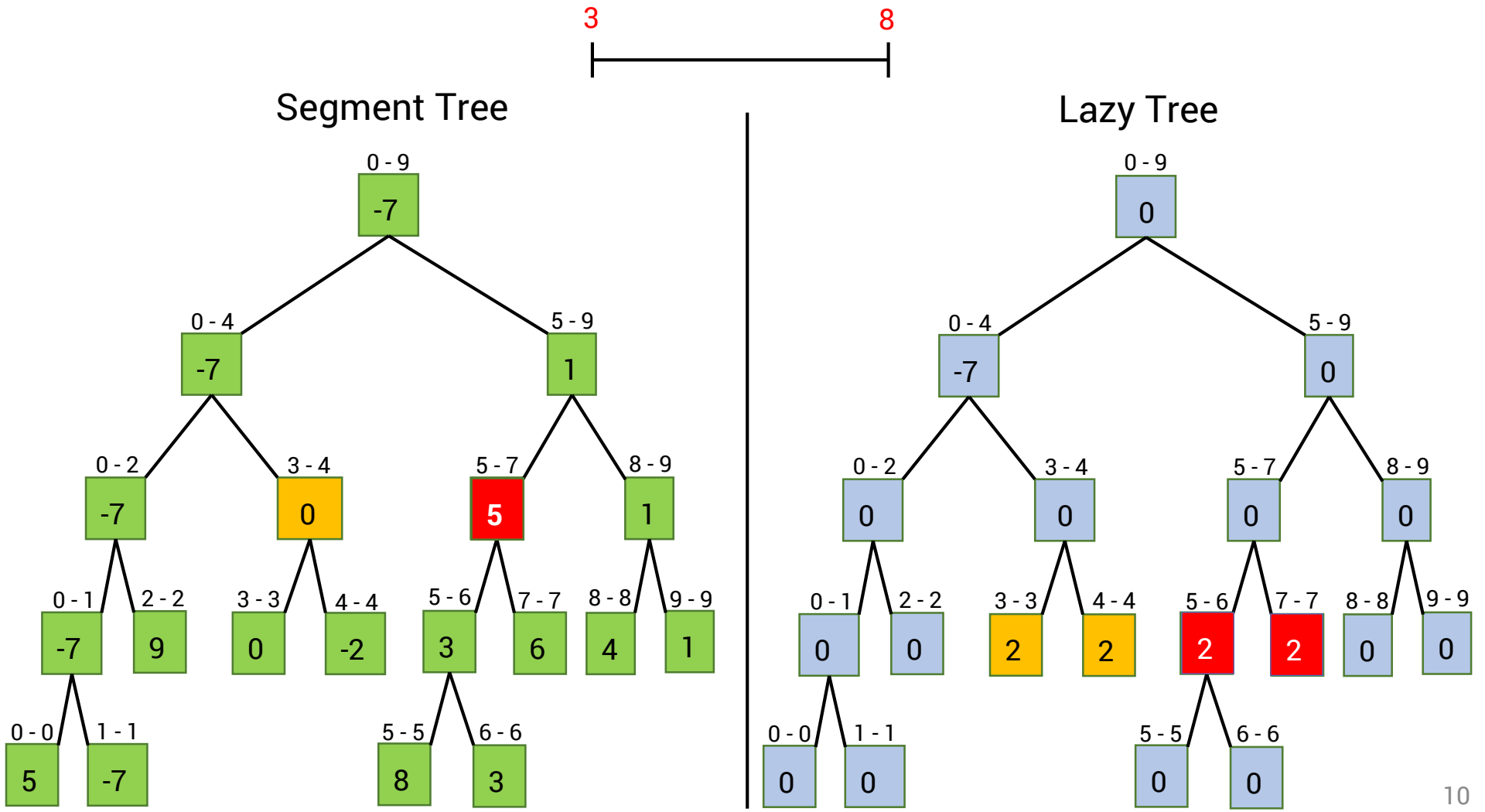
Bước 2: Cập nhật lại node (0, 4)

Giá trị của node (0, 4) được cập nhật lại ➔ Không thay đổi giá trị node này.



Bước 3: Cập nhật node (5, 7)

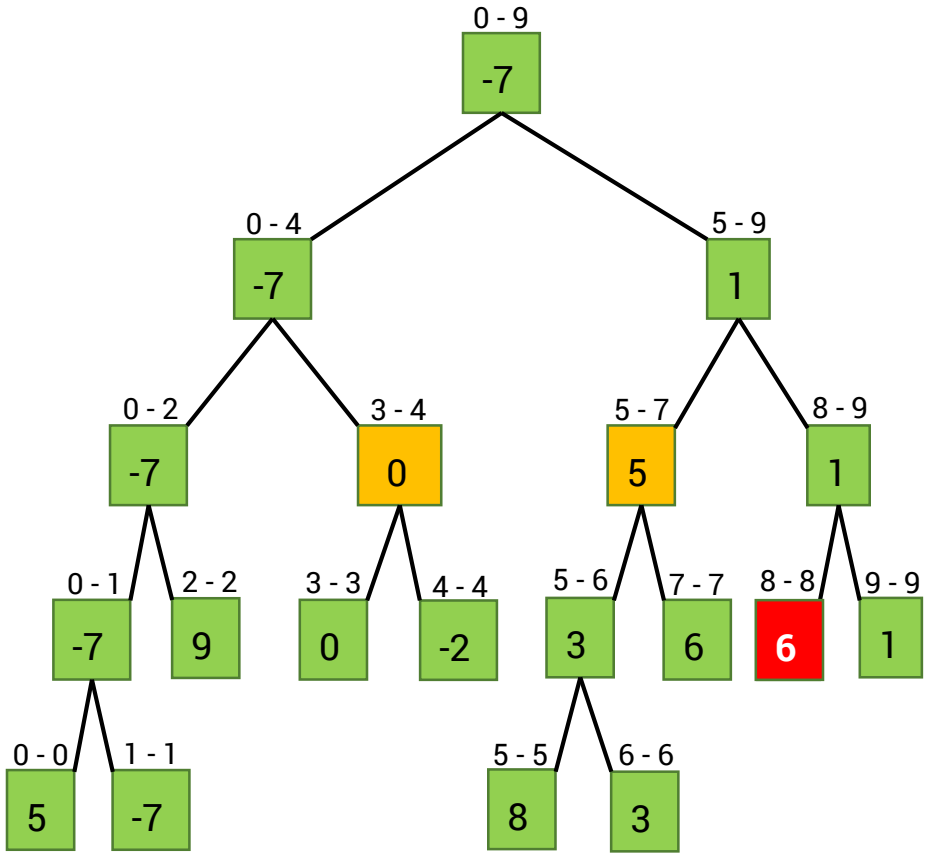
Tìm đến node (5, 7) của Segment Tree để tăng giá trị lên (+2). Sau đó tăng giá trị (+2) trên node con của cây Lazy.



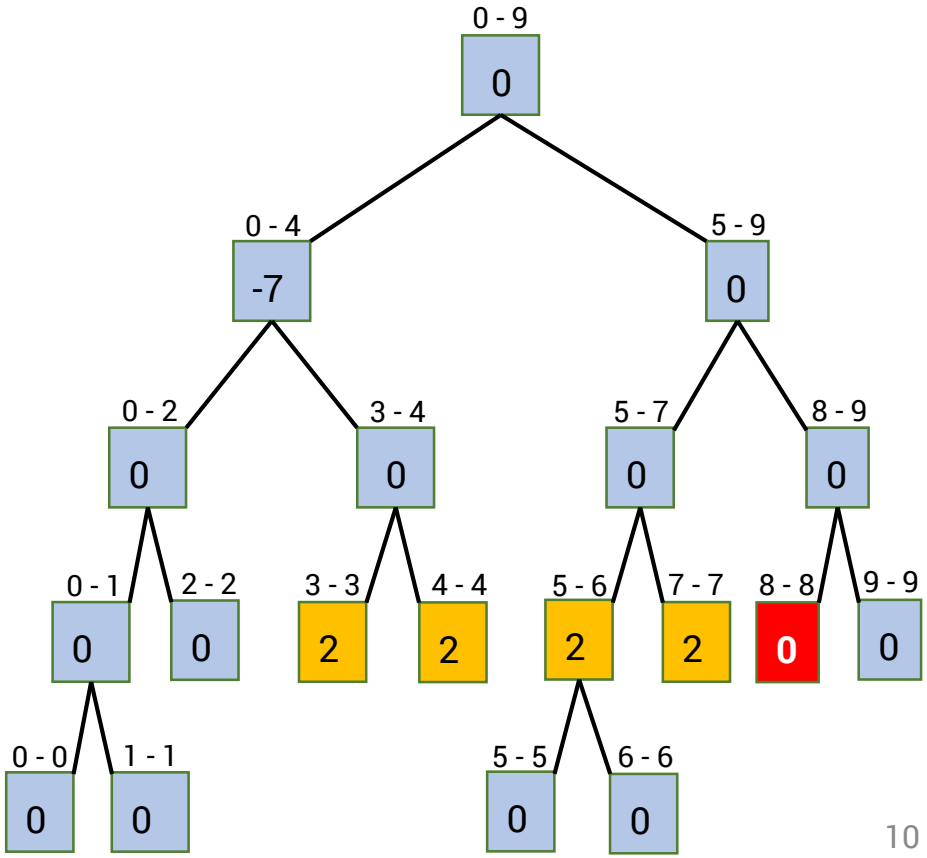
Bước 4: Cập nhật node (8, 8)

Tìm đến node (8, 8) của Segment Tree để tăng giá trị lên (+2). Node trên cây Lazy không có node con nên không thay đổi giá trị ở bước này.

Segment Tree

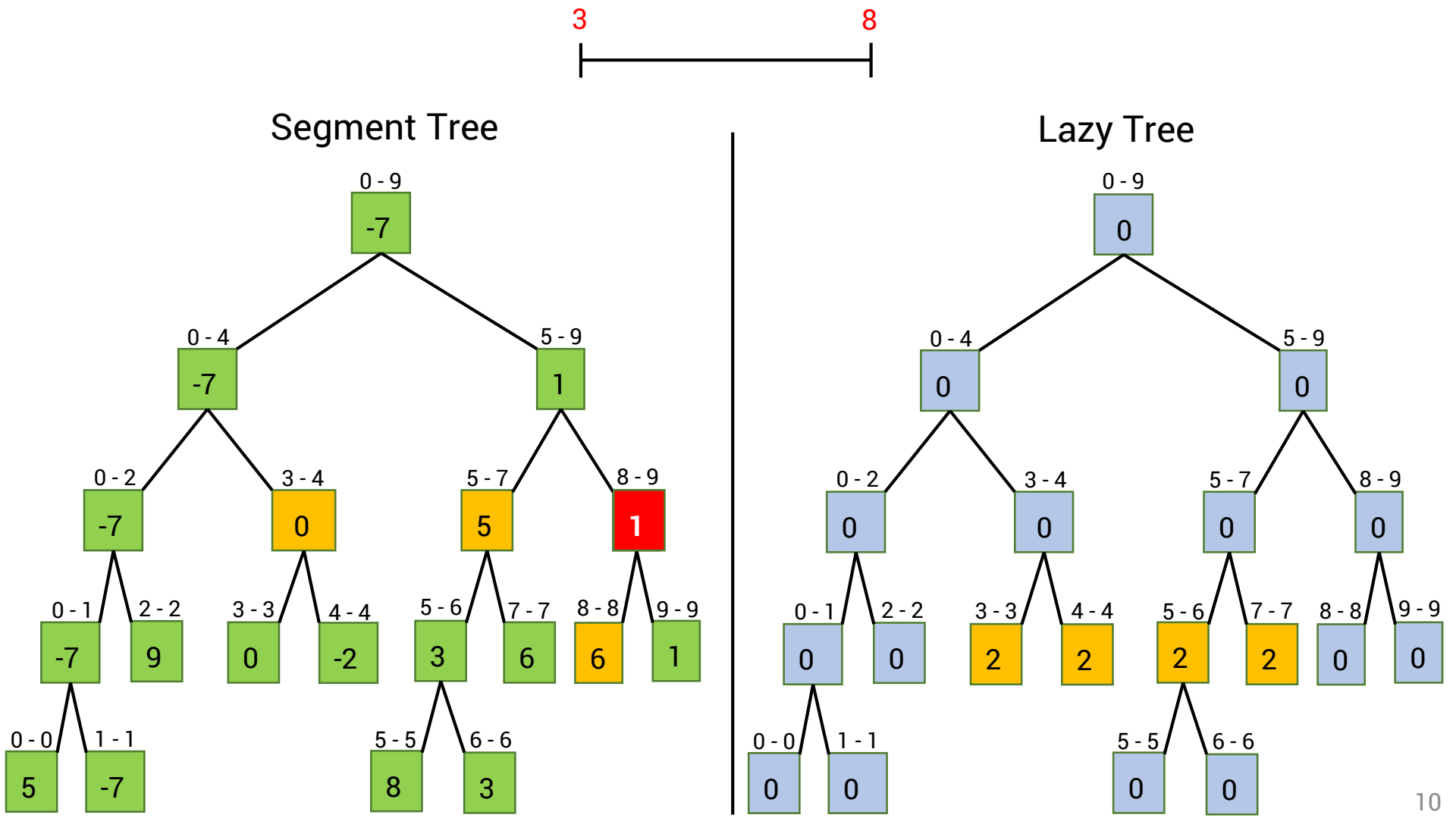


Lazy Tree



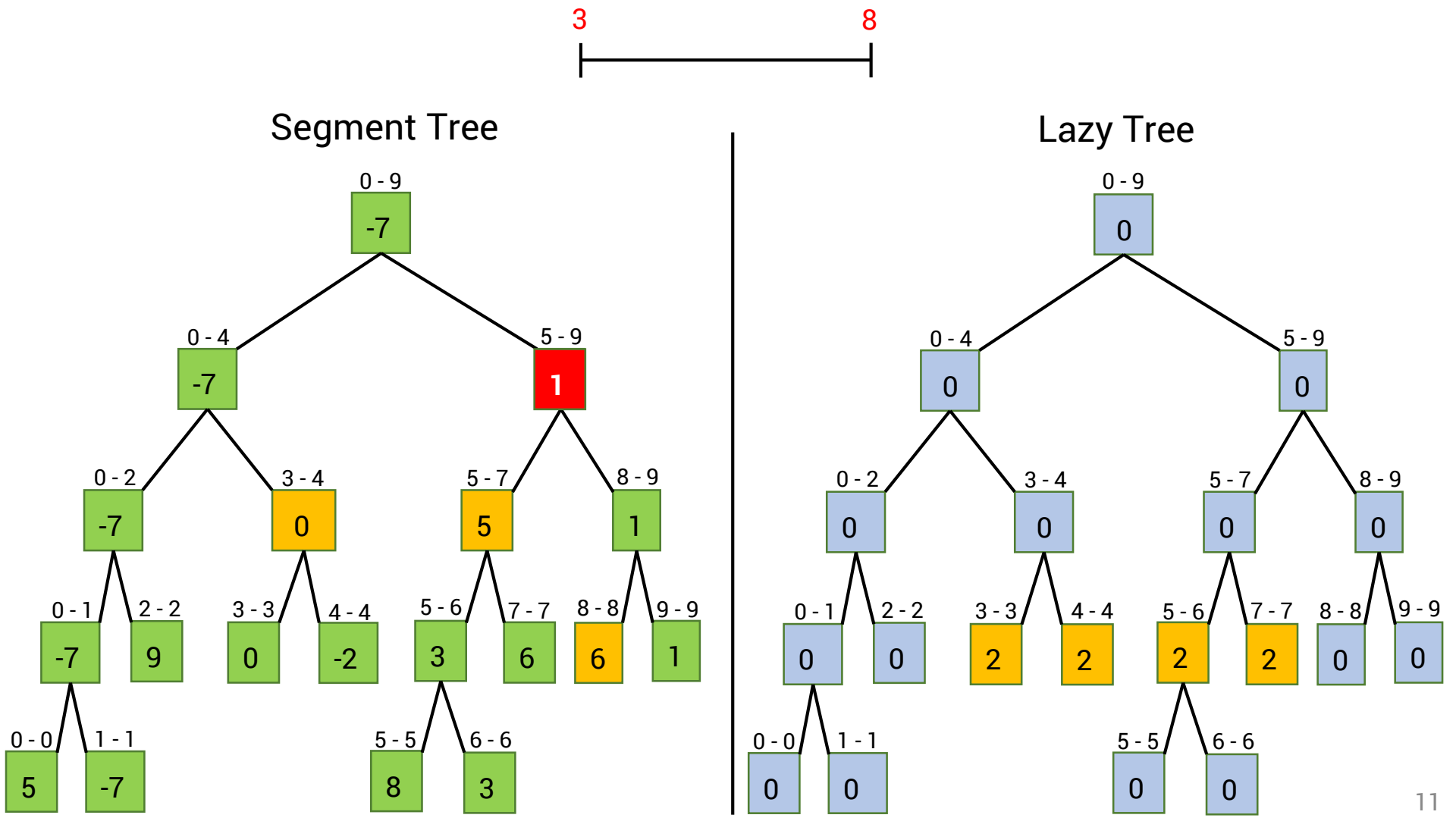
Bước 5: Cập nhật lại node (8, 9)

Giá trị của node (8, 9) được cập nhật lại ➔ Không thay đổi giá trị của node này.



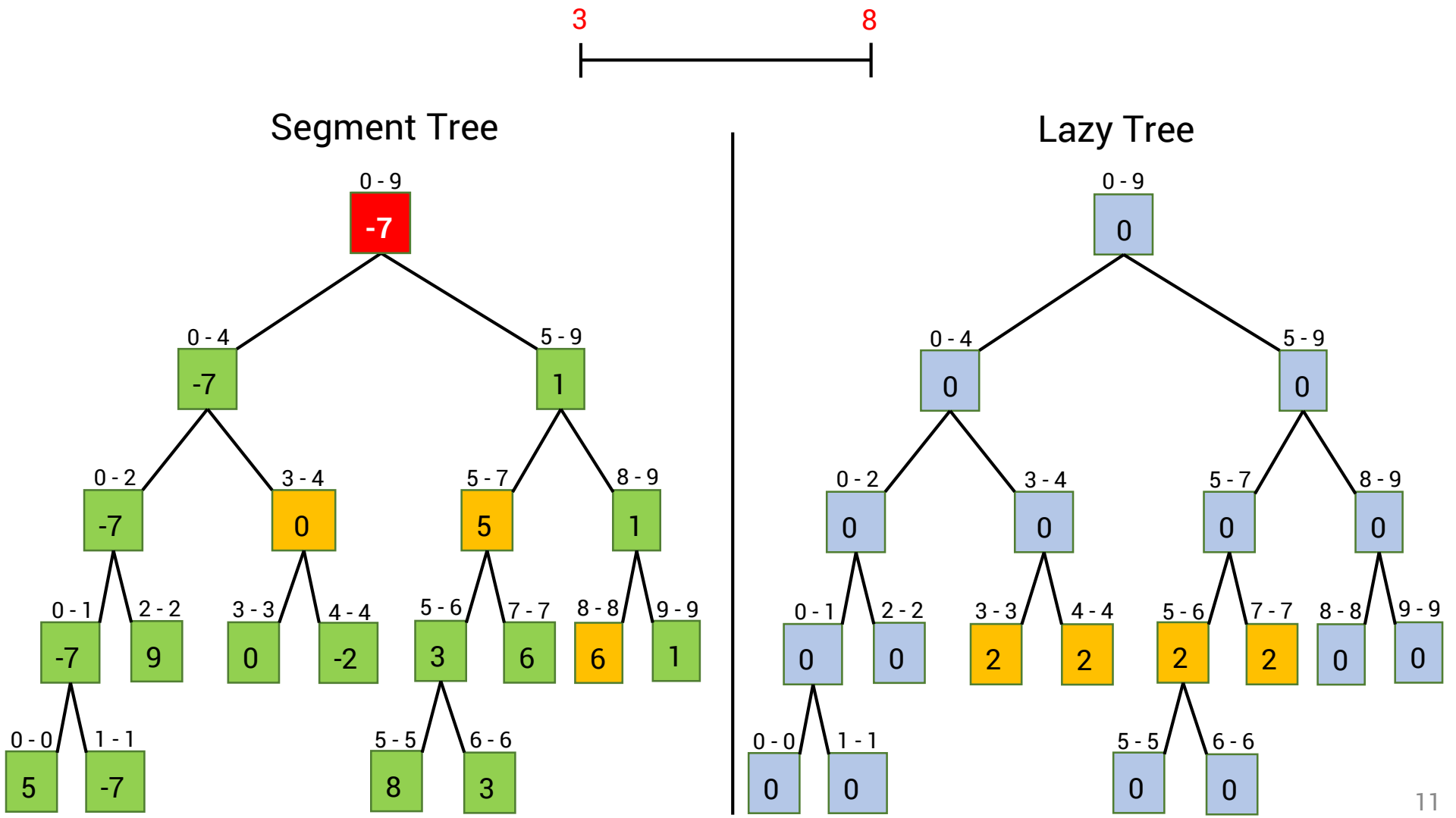
Bước 6: Cập nhật lại node (5, 9)

Giá trị của node (5, 9) được cập nhật lại ➔ Không thay đổi giá trị của node này.



Bước 7: Cập nhật lại node (0, 9)

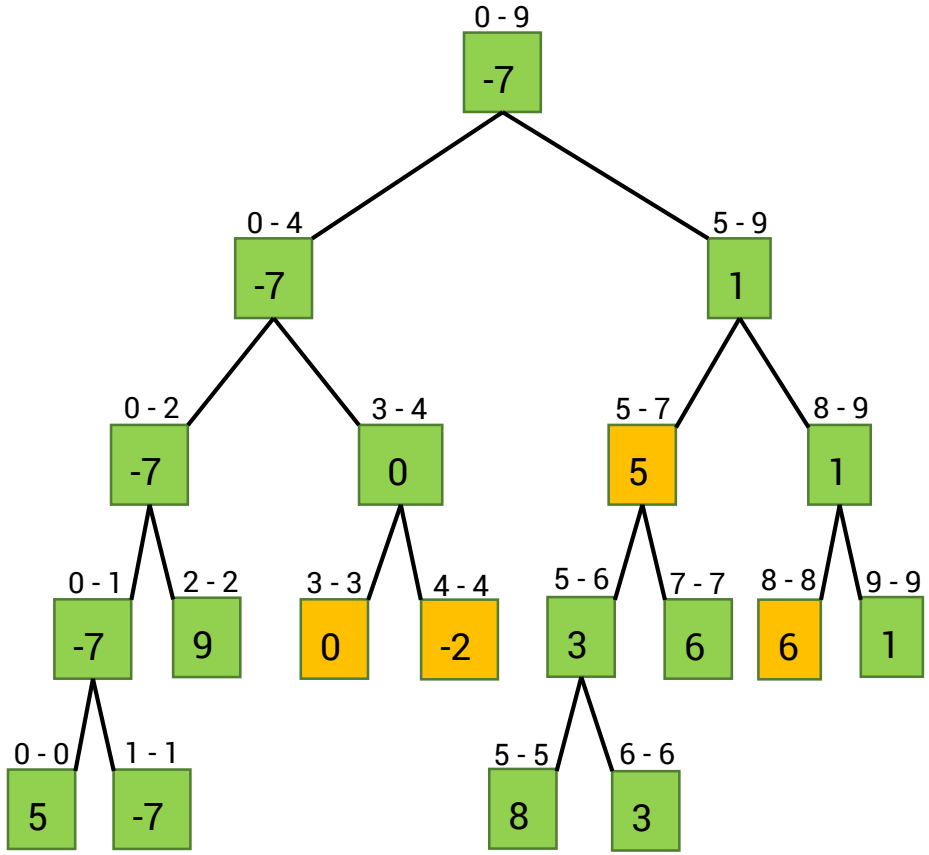
Giá trị của node (0, 9) được cập nhật lại ➔ Không thay đổi giá trị của node này.



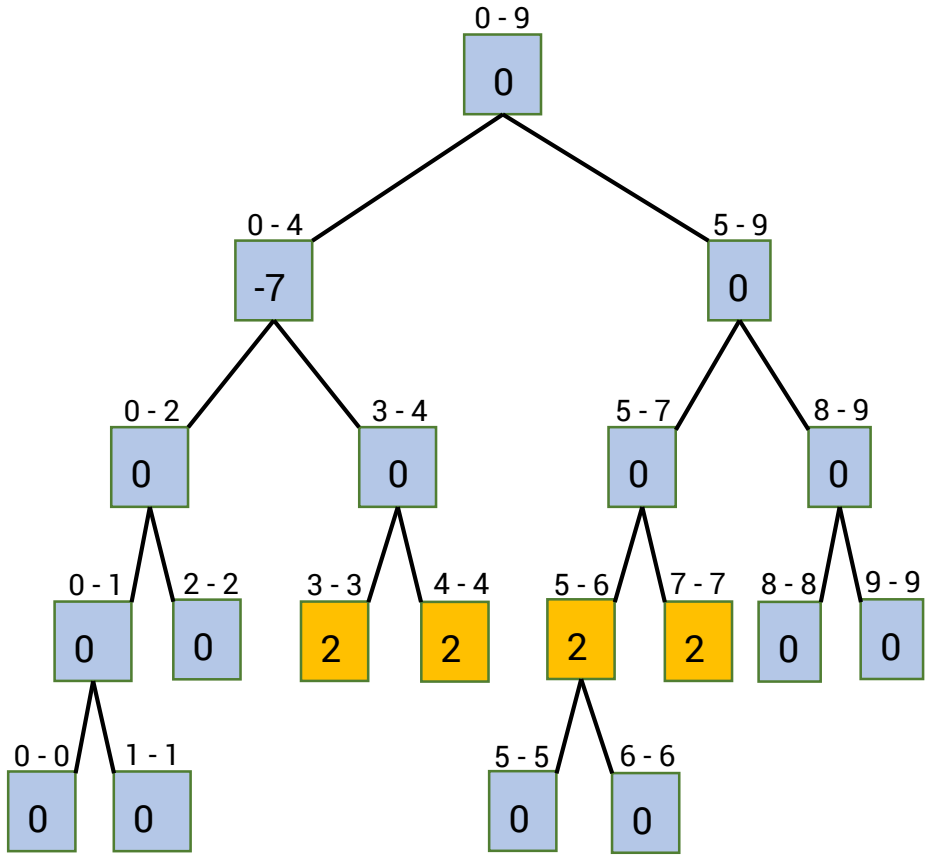
Bước 8: Kết quả lưu trên cây

0	1	2	3	4	5	6	7	8	9
5	-7	9	2	0	10	5	8	6	1

Segment Tree

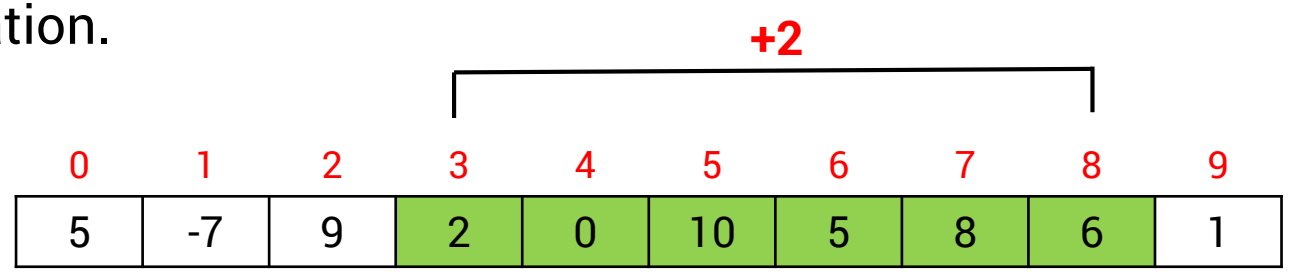


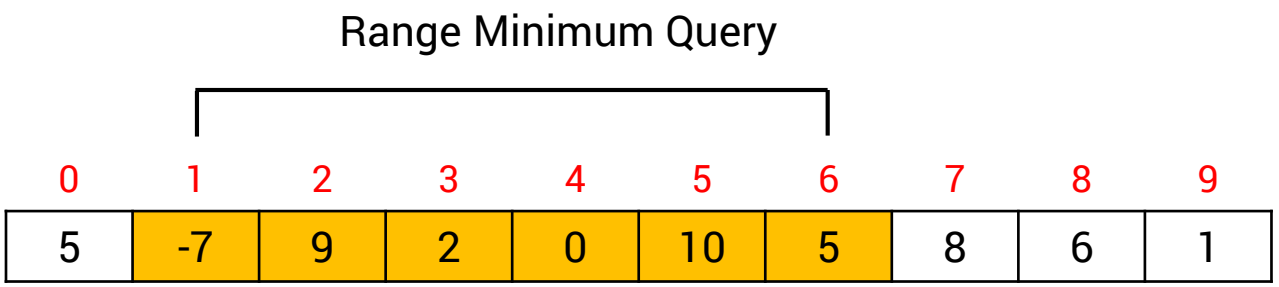
Lazy Tree



Lazy Propagation - Range Minimum Query

Lazy Propagation - RMQ: Cập nhật đoạn bất kỳ delta đơn vị, sau đó tìm giá trị nhỏ nhất của một đoạn bất kỳ trên dãy số với kỹ thuật Lazy Propagation.





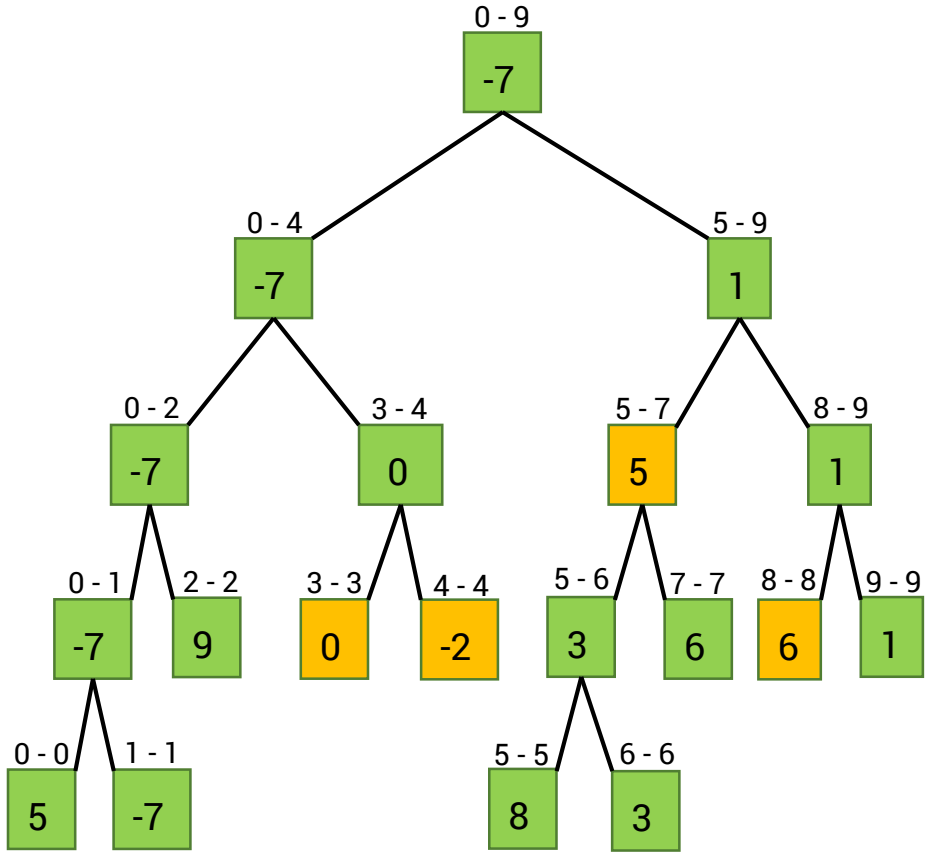
Giá trị nhỏ nhất trong đoạn [1, 6]: -7

Time complexity: O(LogN)

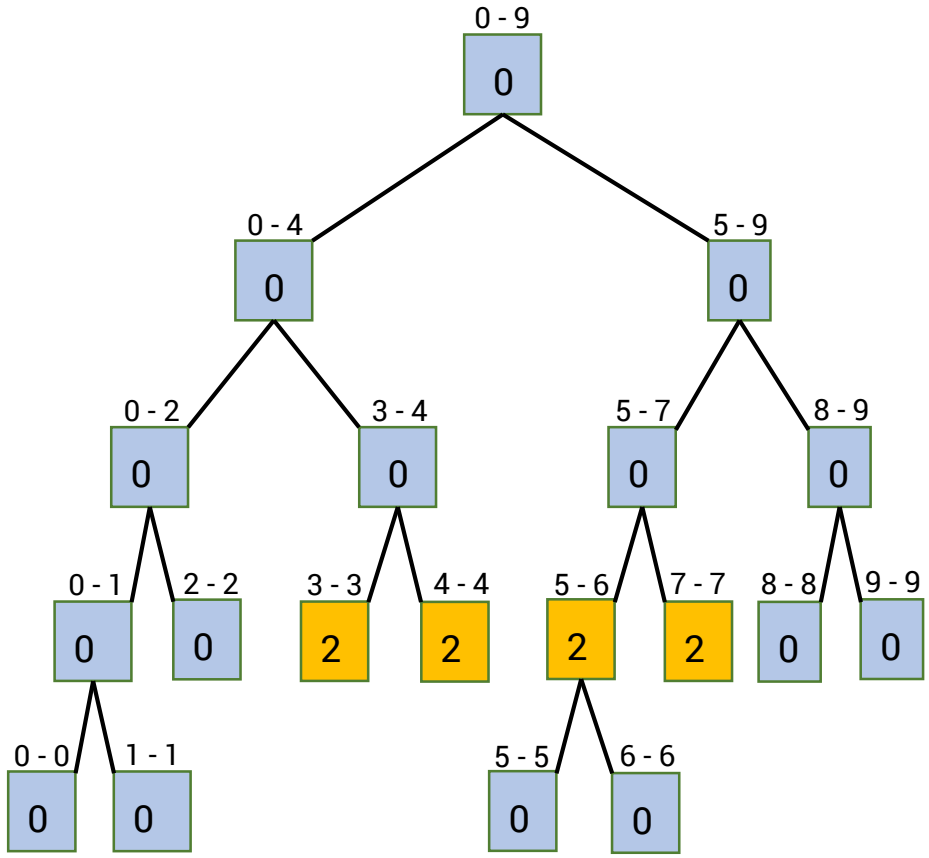
Bước 0: Cây ban đầu

0	1	2	3	4	5	6	7	8	9
5	-7	9	2	0	10	5	8	6	1

Segment Tree

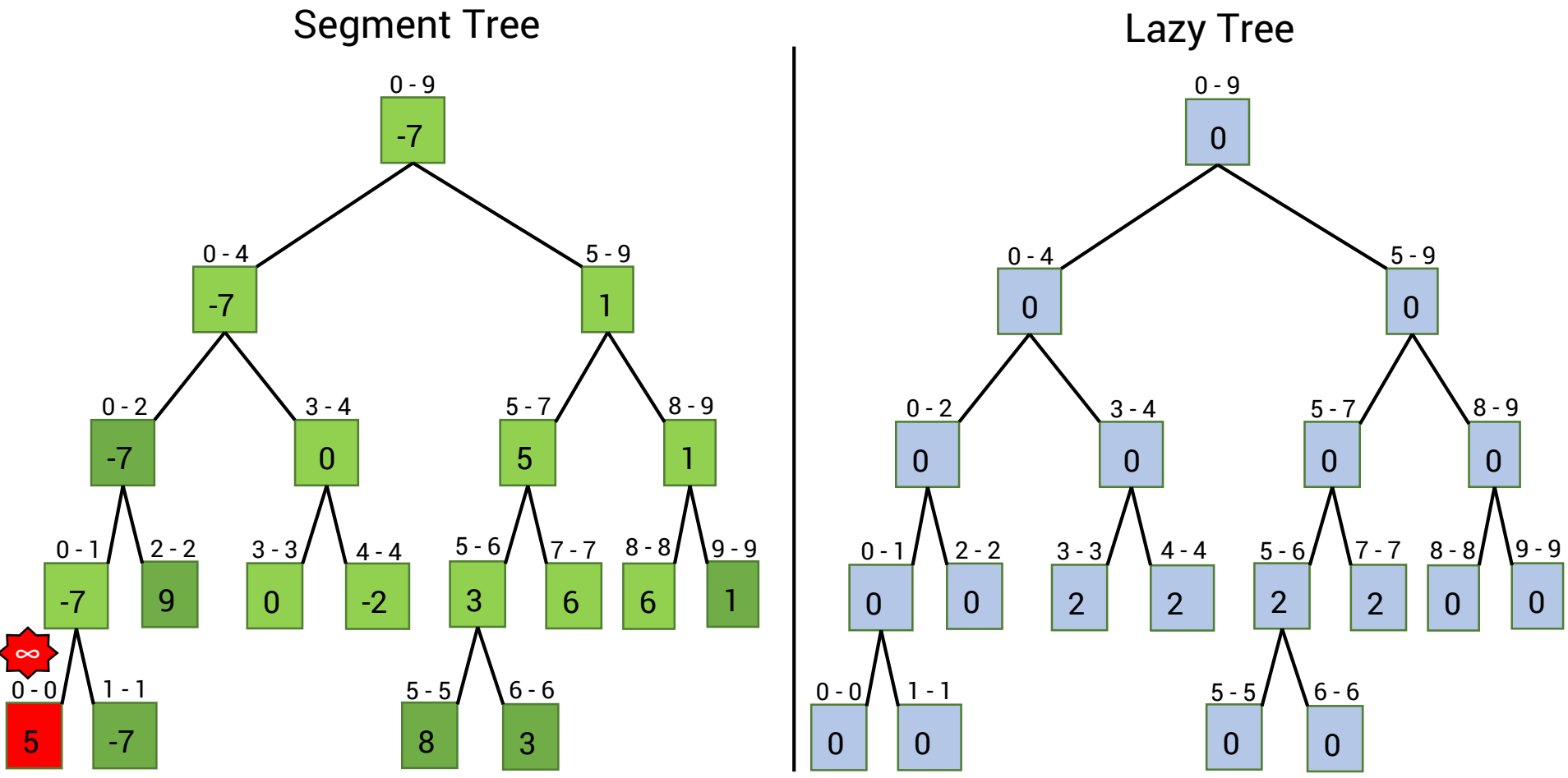


Lazy Tree



Bước 1: Tìm giá trị nhánh 1

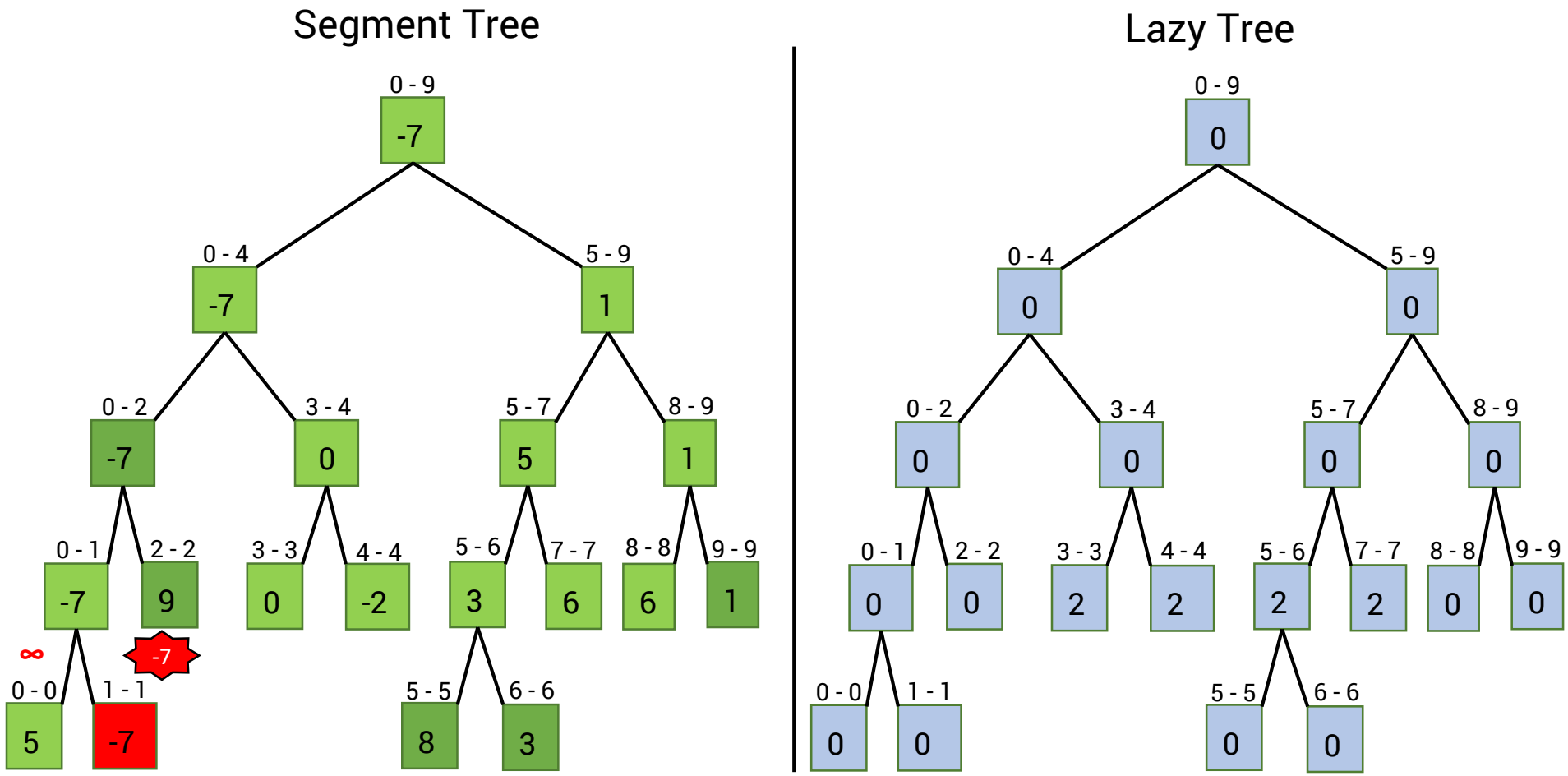
Giá trị nhánh 1, node [0 – 0] nằm **ngoài** đoạn tìm min [1, 6].



Giá trị nhánh 1: ∞

Bước 2: Tìm giá trị nhánh 2

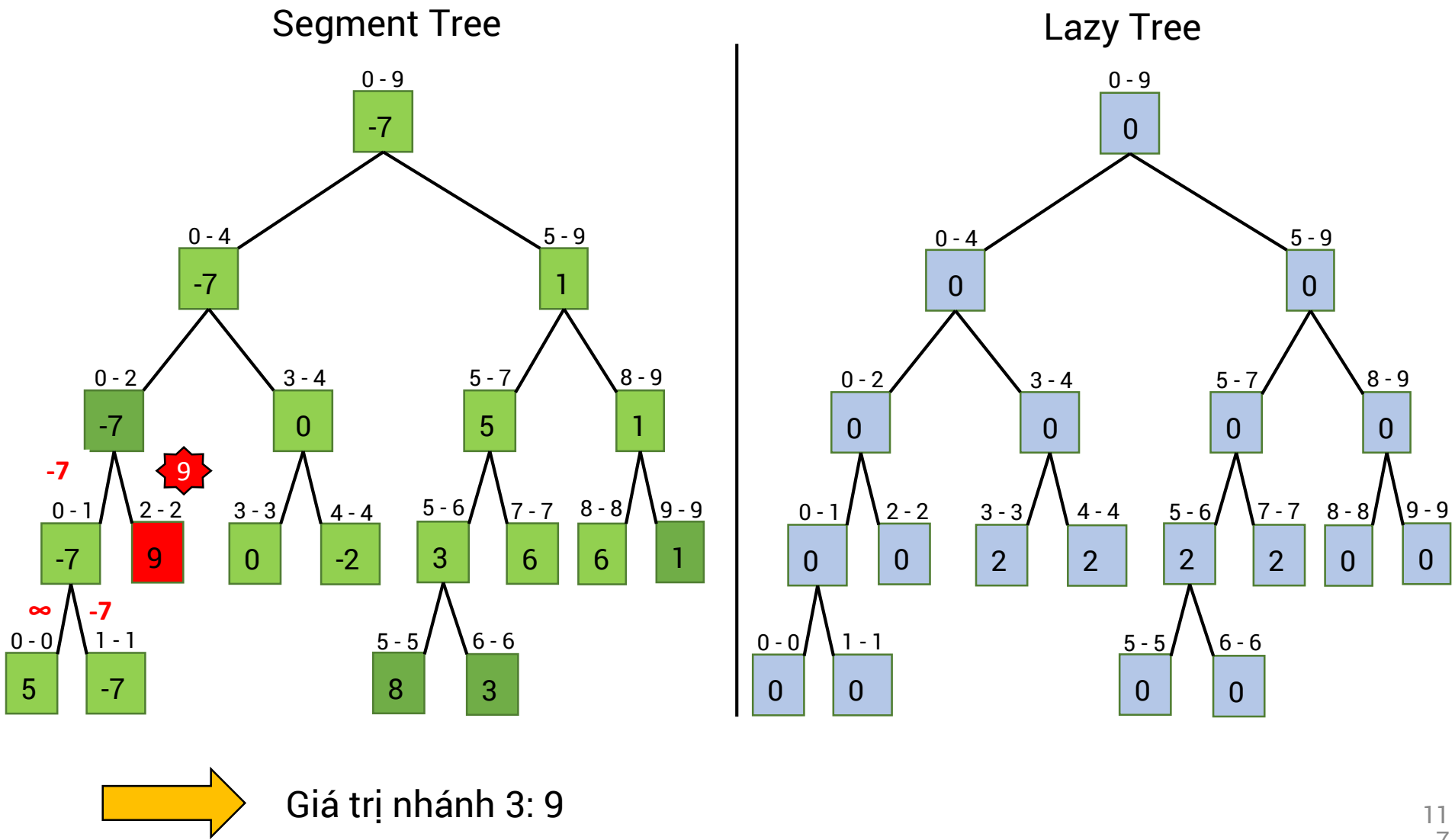
Giá trị nhánh 2, node [1 – 1] nằm trong đoạn tìm min [1, 6].



Giá trị nhánh 2: -7

Bước 3: Tìm giá trị nhánh 3

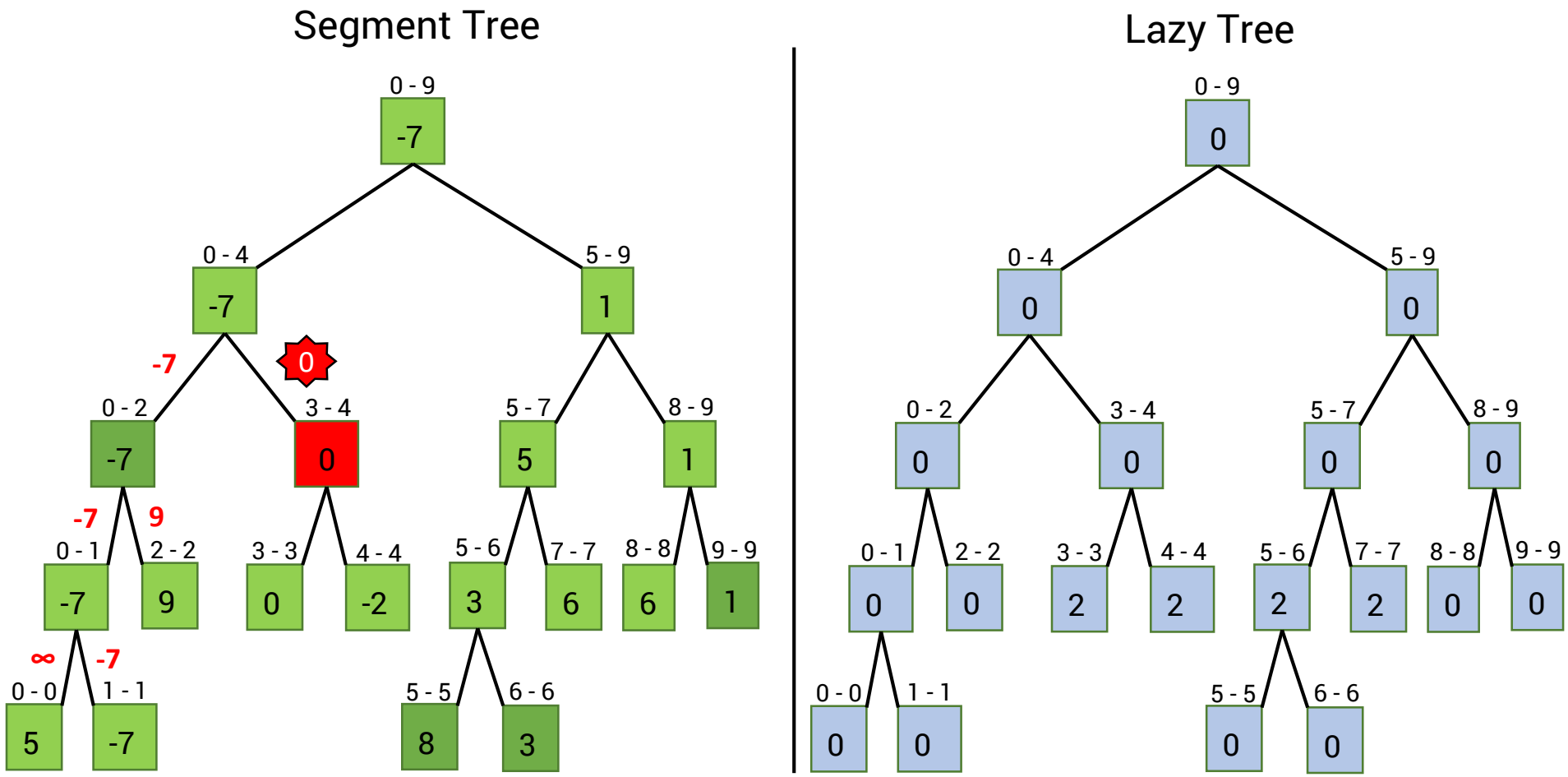
Giá trị nhánh 3, node [2 – 2] nằm trong đoạn tìm min [1, 6].



Giá trị nhánh 3: 9

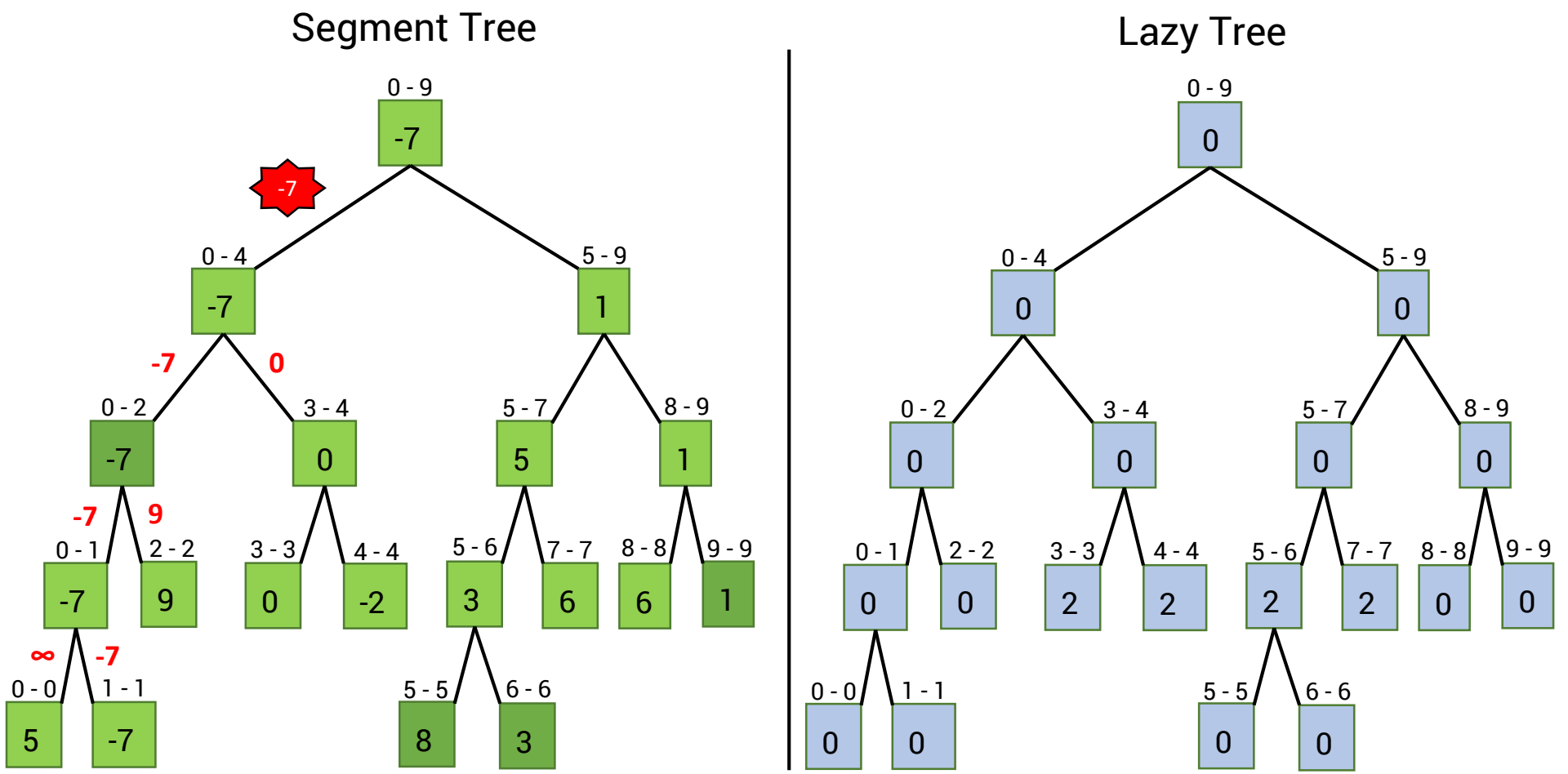
Bước 4: Tìm giá trị nhánh 4

Giá trị nhánh 4, node [3 – 4] nằm trong đoạn tìm min [1, 6].



Giá trị nhánh 4: 0

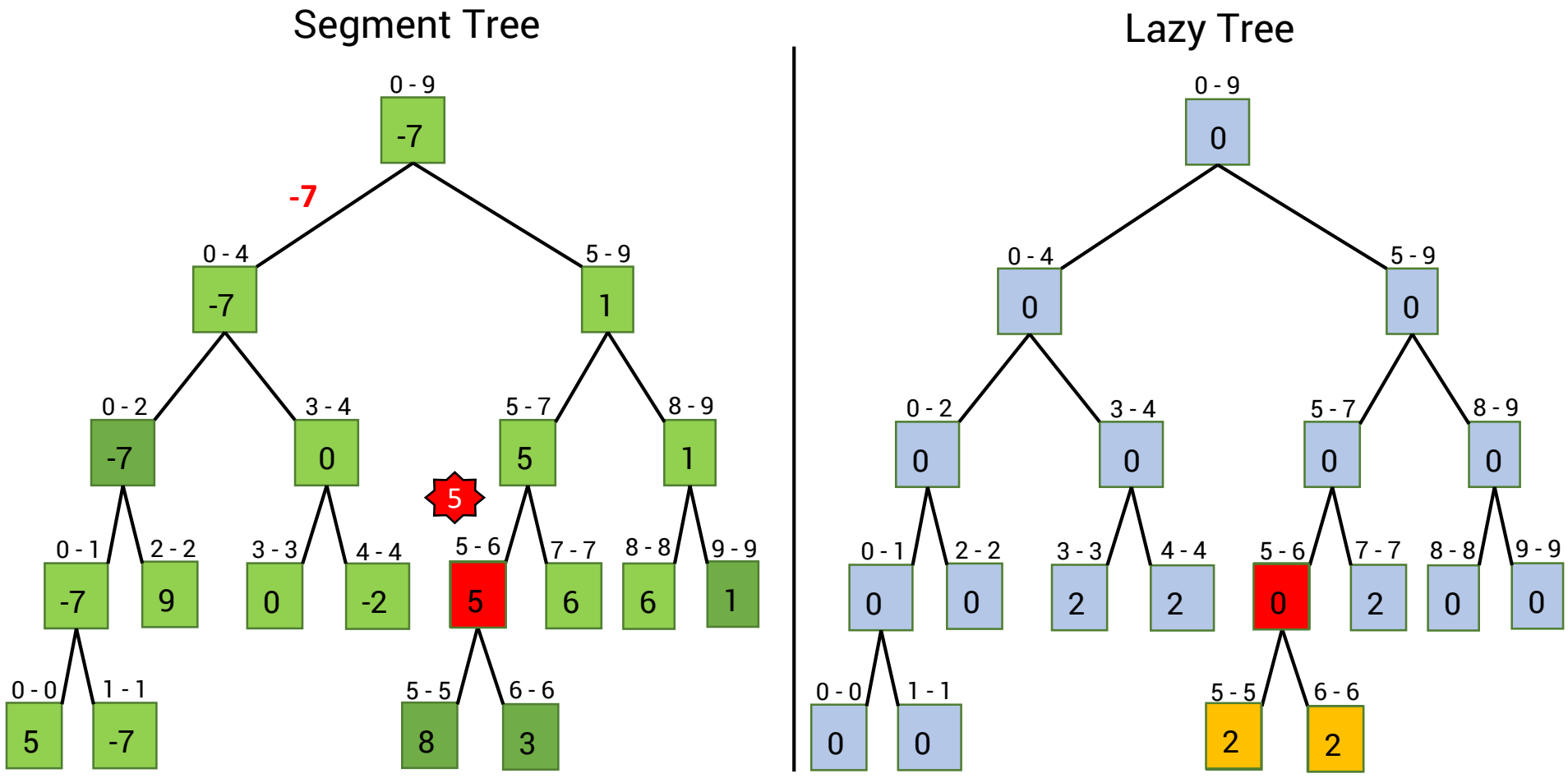
Bước 5: Tìm giá trị toàn bộ nhánh trái



Giá trị nhánh toàn bộ nhánh trái: -7

Bước 6: Tìm giá trị nhánh 5

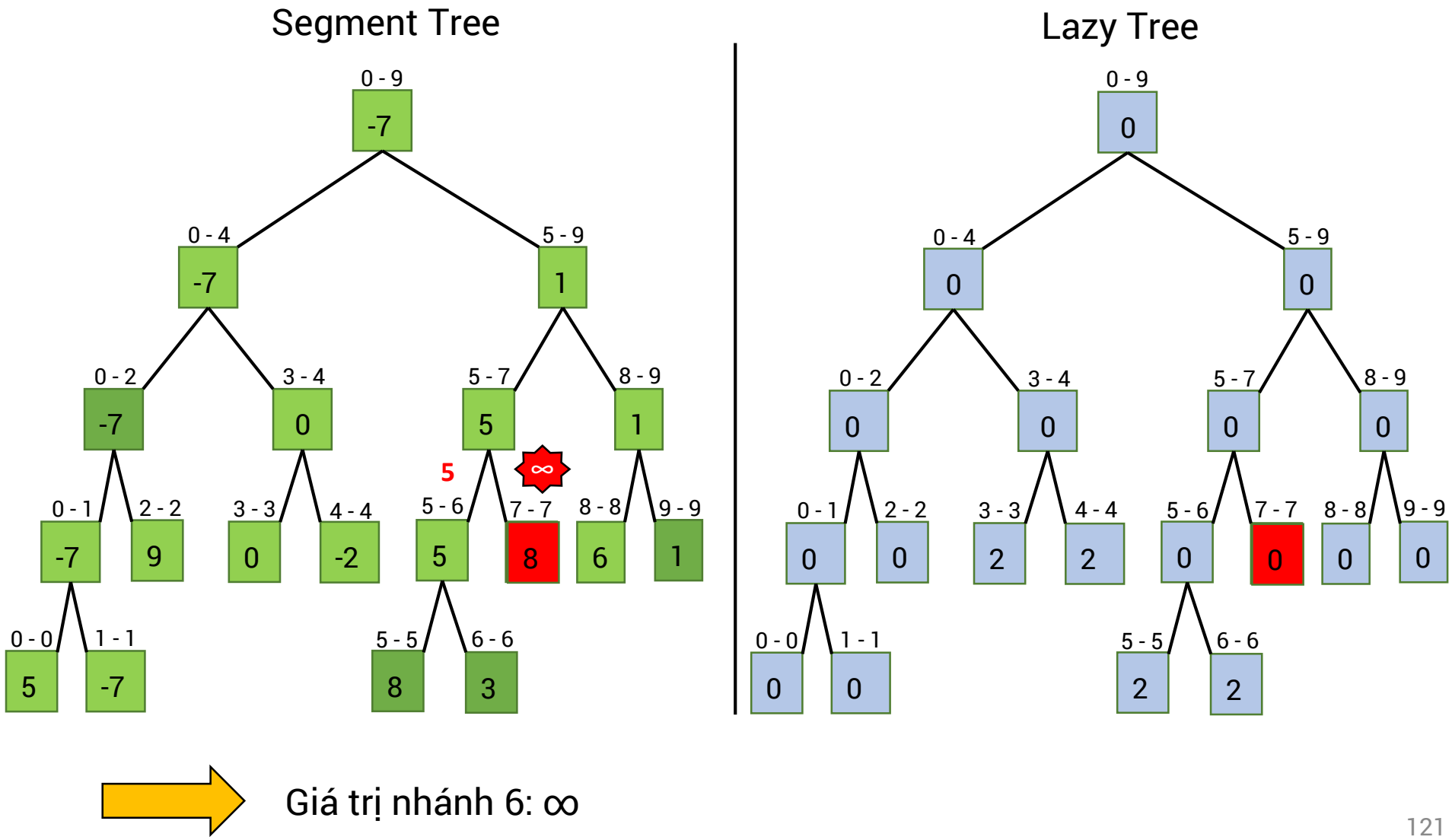
Giá trị nhánh 5, node [5 – 6] nằm trong đoạn tìm min [1, 6].



Giá trị nhánh 5: 5

Bước 7: Tìm giá trị của nhánh 6

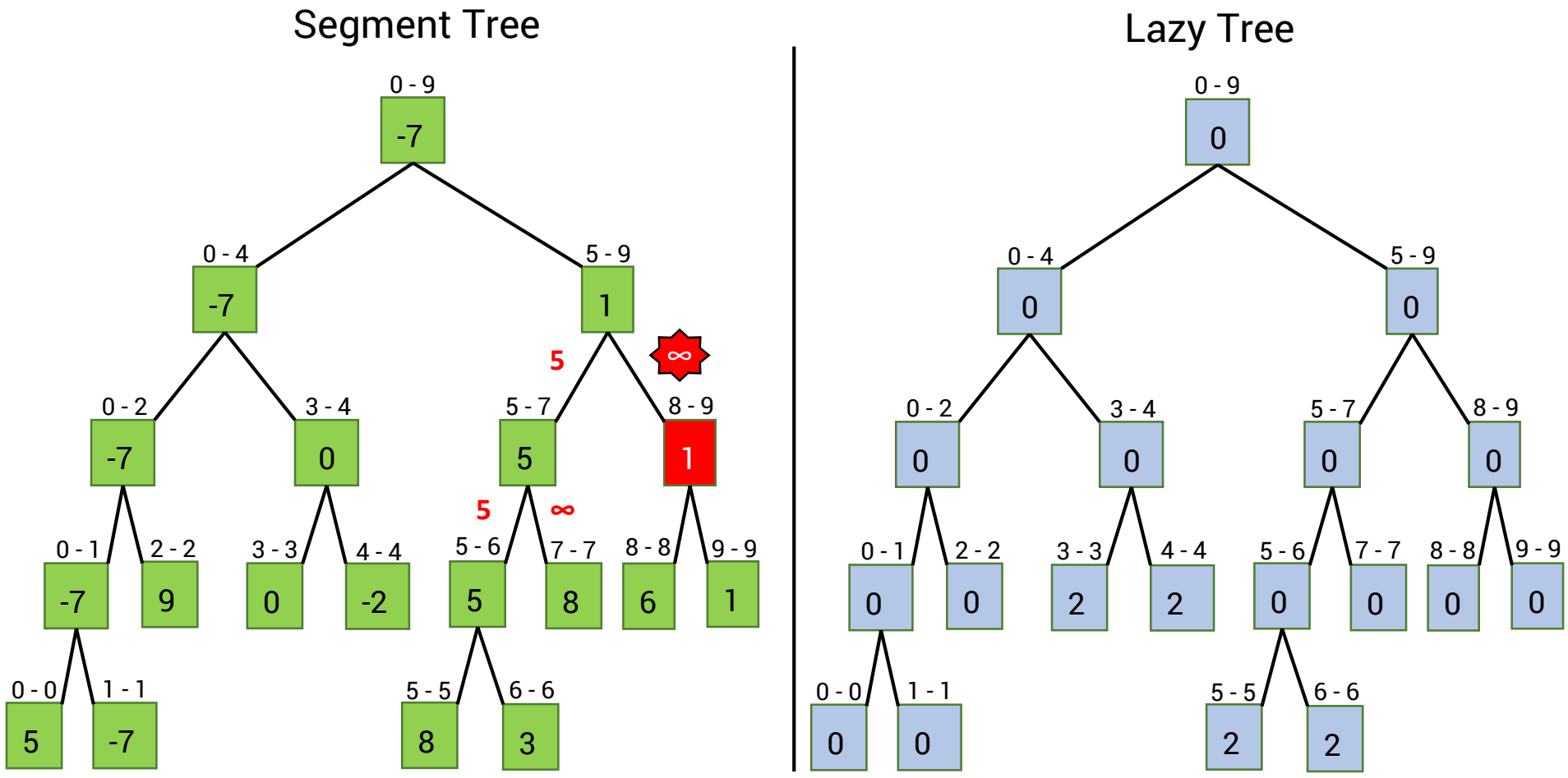
Giá trị nhánh 6, node [7 – 7] nằm **ngoài** đoạn tìm min [1, 6].



Giá trị nhánh 6: ∞

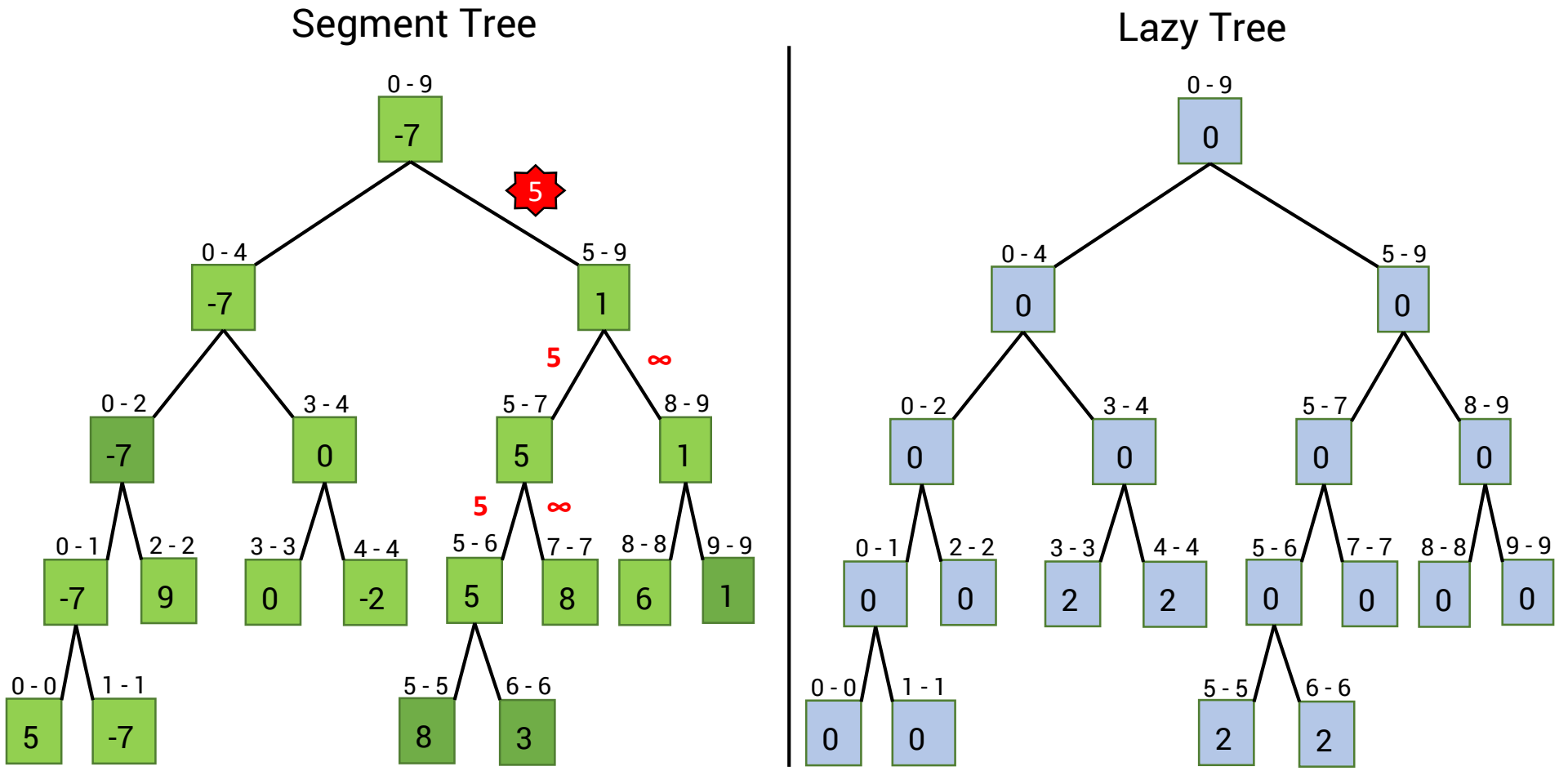
Bước 8: Tìm giá trị của nhánh 7

Giá trị nhánh 7, node [8 – 9] nằm **ngoài** đoạn tìm min [1, 6].



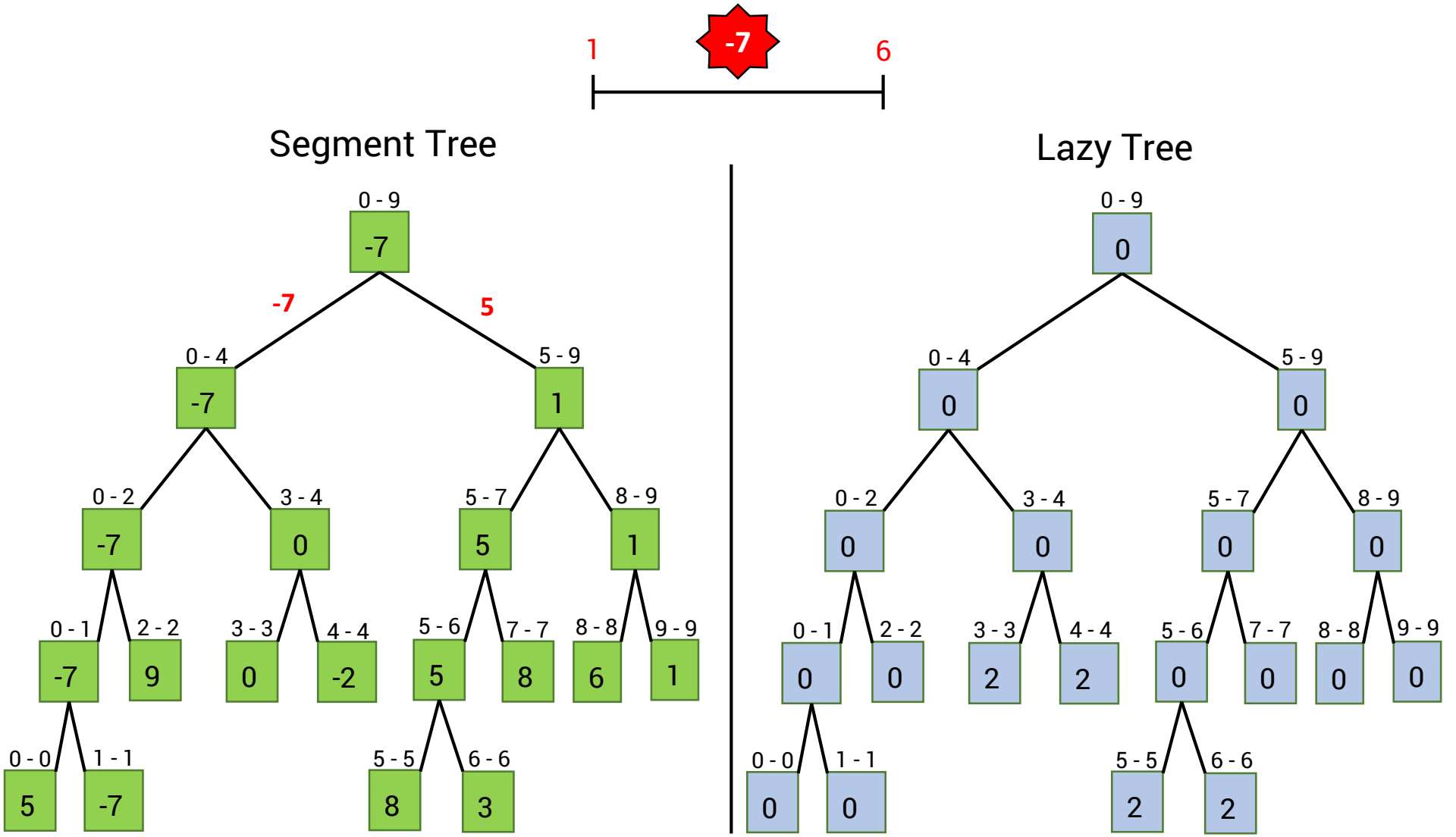
Giá trị nhánh 7: ∞

Bước 9: Tìm giá trị min toàn bộ nhánh phải



Giá trị toàn bộ nhánh phải: 5

Bước 10: Tìm giá trị nhỏ nhất của đoạn [1, 6]



Giá trị nhỏ nhất đoạn [1, 6]: -7

Source Code Lazy Propagation - RQM


```cpp
#include <iostream>
#include <algorithm>
#include <cmath>
#include <vector>
using namespace std;

#define INF 1e9

void buildTree(vector<int> &a, vector<int> &segtree, int left, int right, int index) {
    if (left == right) {
        segtree[index] = a[left];
        return;
    }
    int mid = (left + right) / 2;

    buildTree(a, segtree, left, mid, 2 * index + 1);
    buildTree(a, segtree, mid + 1, right, 2 * index + 2);

    segtree[index] = min(segtree[2 * index + 1], segtree[2 * index + 2]);
}
```

Source Code Lazy Propagation - RQM

```
21. void updateQuery_minRangeLazy(vector<int> &segtree, vector<int> &lazy, int left,
                                  int right, int from, int to, int delta, int index) {
22.     if (left > right) {
23.         return;
24.     }
25.     //make sure all propagation is done at index. If not update tree
26.     //at index and mark its children for lazy propagation.
27.     if (lazy[index] != 0) {
28.         segtree[index] += lazy[index];
29.         if (left != right) { //not a leaf node
30.             lazy[2 * index + 1] += lazy[index];
31.             lazy[2 * index + 2] += lazy[index];
32.         }
33.         lazy[index] = 0;
34.     }
35.     //no overlap condition
36.     if (from > right || to < left) {
37.         return;
38.     }
```

Source Code Lazy Propagation - RQM

```
39.       //total overlap condition
40.       if (from <= left && to >= right)
41.       {
42.           segtree[index] += delta;
43.           if (left != right) {
44.               lazy[2 * index + 1] += delta;
45.               lazy[2 * index + 2] += delta;
46.           }
47.           return;
48.       }
49.       int mid = (left + right) / 2;
50.       updateQuery_minRangeLazy(segtree, lazy, left, mid, from, to, delta, 2 * index + 1);
51.       updateQuery_minRangeLazy(segtree, lazy, mid + 1, right, from, to, delta, 2 * index + 2);
52.       segtree[index] = min(segtree[2 * index + 1], segtree[2 * index + 2]);
53.   }
```

Source Code Lazy Propagation - RQM

```
53. int minRangeLazy(vector<int> &segtree, vector<int> &lazy, int left, int right,
                     int from, int to, int index) {
54.     if (left > right) {
55.         return INF;
56.     }
57.     if (lazy[index] != 0) {
58.         segtree[index] += lazy[index];
59.         if (left != right) { //not a leaf node
60.             lazy[2 * index + 1] += lazy[index];
61.             lazy[2 * index + 2] += lazy[index];
62.         }
63.         lazy[index] = 0;
64.     }
65.
66.     //no overlap
67.     if (from > right || to < left) {
68.         return INF;
69.     }
```

Source Code Lazy Propagation - RQM

```
70.       //total overlap
71.       if (from <= left && to >= right) {
72.           return segtree[index];
73.       }
74.       //partial overlap
75.       int mid = (left + right) / 2;
76.       return min(minRangeLazy(segtree, lazy, mid + 1, right, from, to, 2 * index + 2),
                     minRangeLazy(segtree, lazy, left, mid, from, to, 2 * index + 1));
77.   }
```

Source Code Lazy Propagation - RQM


```
78. int main()
79. {
80.     vector<int> a = { 5, -7, 9, 0, -2, 8, 3, 6, 4, 1 };
81.     int n = a.size();
82.      //Height of segment tree
83.     int h = (int)ceil(log2(n));
84.     //Maximum size of segment tree
85.     int sizetree = 2 * (int)pow(2, h) - 1;
86.     vector<int> segtree(sizetree, INF);
87.     vector<int> lazy(sizetree, 0);
88.
89.     buildTree(a, segtree, 0, n - 1, 0);
90.
91.     int fromindex = 3;
92.     int toindex = 8;
93.
94.     //increment [3, 8] by 2.
95.     int delta = 2;
96.
97.     updateQuery_minRangeLazy(segtree, lazy, 0, n - 1, fromindex, toindex, delta, 0);
```

Source Code Lazy Propagation - RQM

```
98.         updateQuery_minRangeLazy(segtree, lazy, 0, n - 1, fromindex, toindex, delta, 0);
99.         fromindex = 1;
100.        toindex = 6;
101.        int res = minRangeLazy(segtree, lazy, 0, n - 1, fromindex, toindex, 0);
102.        cout << res << endl;
103.        return 0;
104.    }
```

Source Code Lazy Propagation - RQM

```python
from math import ceil, log2
INF = 10**9
def updateQuery_minRangeLazy(segtree, lazy, left, right, fr, to, delta, index):
    if left > right:
        return
    if lazy[index] != 0:
        segtree[index] += lazy[index]
        if left != right:
            lazy[2 * index + 1] += lazy[index]
            lazy[2 * index + 2] += lazy[index]
        lazy[index] = 0
      # no overlap condition
    if fr > right or to < left:
        return
     # total overlap condition
    if fr <= left and right <= to:
        segtree[index] += delta
        if left != right:
            lazy[2 * index + 1] += delta
            lazy[2 * index + 2] += delta
        return
```

Source Code Lazy Propagation - RQM

```
22.     # otherwise partial overlap so look both left and right
23.     mid = (left + right) // 2
24.     updateQuery_minRangeLazy(segtree, lazy, left, mid, fr, to, delta, 2 * index + 1)
25.     updateQuery_minRangeLazy(segtree, lazy, mid + 1, right, fr, to, delta, 2 * index + 2)
26.     segtree[index] = min(segtree[2 * index + 1], segtree[2 * index + 2])
```

Source Code Lazy Propagation - RQM

Big-O Orange

```python
def minRangeLazy(segtree, lazy, left, right, fr, to, index):
    if left > right:
        return INF
    if lazy[index] != 0:
        segtree[index] += lazy[index]
        if left != right: # not a leaf node
            lazy[2 * index + 1] += lazy[index]
            lazy[2 * index + 2] += lazy[index]
        lazy[index] = 0
    # no overlap
    if fr > right or to < left:
        return INF
    # total overlap
    if fr <= left and to >= right:
        return segtree[index]

    # partial overlap
    mid = (left + right) // 2
    return min(minRangeLazy(segtree, lazy, mid + 1, right, fr, to, 2 * index + 2),
               minRangeLazy(segtree, lazy, left, mid, fr, to, 2 * index + 1))
```

Source Code Lazy Propagation - RQM

```
47.  def buildTree(a, segtree, left, right, index):
48.      if left == right:
49.          segtree[index] = a[left]
50.          return
51.      mid = (left + right) // 2
52.      buildTree(a, segtree, left, mid, 2 * index + 1)
53.      buildTree(a, segtree, mid + 1, right, 2 * index + 2)
54.      segtree[index] = min(segtree[2 * index + 1], segtree[2 * index + 2])
```

Source Code Lazy Propagation - RQM

Big-O Orange

```
55. if __name__ == '__main__':
56.     a = [5, -7, 9, 0, -2, 8, 3, 6, 4, 1]
57.     n = len(a)
58.     h = ceil(log2(n))
59.     sizeTree = 2 * (2**h) - 1
60.     segtree = [INF] * sizeTree
61.     lazy = [0] * sizeTree
62.     buildTree(a, segtree, 0, n - 1, 0)
63.     fromindex = 3
64.     toindex = 8
65.     # increase [3, 8] by 2
66.     delta = 2
67.     updateQuery_minRangeLazy(segtree, lazy, 0, n - 1, fromindex, toindex, delta, 0)
68.
69.     fromindex = 1
70.     toindex = 6
71.     res = minRangeLazy(segtree, lazy, 0, n - 1, fromindex, toindex, 0)
72.     print(res)
```

Source Code Lazy Propagation - RQM

Big-O Orange


```java
import java.util.Arrays;

public class Main {
    private static final int INF = (int)1e9;

    private static double log2(int number) {
        return Math.log(number) / Math.log(2);
    }
    private static void buildTree(int[] a, int[] segtree, int left, int right, int index) {
        if (left == right) {
            segtree[index] = a[left];
            return;
        }
        int mid = (left + right) / 2;

        buildTree(a, segtree, left, mid, 2 * index + 1);
        buildTree(a, segtree, mid + 1, right, 2 * index + 2);

        segtree[index] = segtree[2 * index + 1] + segtree[2 * index + 2];
    }
```

Source Code Lazy Propagation - RQM

```
21.     private static void updateQuery_minRangeLazy(int[] segtree, int[] lazy, int left, int right,
        int from, int to, int delta, int index) {
22.         if (left > right) {
23.             return;
24.         }
25.         if (lazy[index] != 0) {
26.             segtree[index] += lazy[index];
27.             if (left != right) { //not a leaf node
28.                 lazy[2 * index + 1] += lazy[index];
29.                 lazy[2 * index + 2] += lazy[index];
30.             }
31.             lazy[index] = 0;
32.         }
33.
34.         //no overlap condition
35.         if (from > right || to < left) {
36.             return;
37.         }
```

Source Code Lazy Propagation - RQM

```java
        //total overlap condition
        if (from <= left && to >= right) {
            segtree[index] += delta;
            if (left != right) {
                lazy[2 * index + 1] += delta;
                lazy[2 * index + 2] += delta;
            }
            return;
        }

        //otherwise partial overlap so look both left and right.
        int mid = (left + right) / 2;
        updateQuery_minRangeLazy(segtree, lazy, left, mid, from, to, delta, 2 * index + 1);
        updateQuery_minRangeLazy(segtree, lazy, mid + 1, right, from, to, delta, 2 * index + 2);
        segtree[index] = Math.min(segtree[2 * index + 1], segtree[2 * index + 2]);
    }
```

Source Code Lazy Propagation - RQM

```
54.     private static int minRangeLazy(int[] segtree, int[] lazy, int left, int right, int from,
    int to, int index) {
55.         if (left > right) {
56.             return INF;
57.         }
58.         if (lazy[index] != 0) {
59.             segtree[index] += lazy[index];
60.             if (left != right) { //not a leaf node
61.                 lazy[2 * index + 1] += lazy[index];
62.                 lazy[2 * index + 2] += lazy[index];
63.             }
64.             lazy[index] = 0;
65.         }
66.
67.         //no overlap
68.         if (from > right || to < left) {
69.             return INF;
70.         }
```

Source Code Lazy Propagation - RQM

```
71.         //total overlap
72.         if (from <= left && to >= right) {
73.             return segtree[index];
74.         }
75.
76.         //partial overlap
77.         int mid = (left + right) / 2;
78.         return Math.min(minRangeLazy(segtree, lazy, mid + 1, right, from, to, 2 * index + 2),
79.                 minRangeLazy(segtree, lazy, left, mid, from, to, 2 * index + 1));
80.     }
```

Source Code Lazy Propagation - RQM

```
81.     public static void main(String[] args) {
82.         int[] a = new int[]{ 5, -7, 9, 0, -2, 8, 3, 6, 4, 1 };
83.         int n = a.length;
84.
85.         //Height of segment tree
86.         int h = (int)Math.ceil(log2(n));
87.
88.         //Maximum size of segment tree
89.         int sizeTree = 2 * (int)Math.pow(2, h) - 1;
90.
91.         int[] segtree = new int[sizeTree];
92.         int[] lazy = new int[sizeTree];
93.         Arrays.fill(segtree, INF);
94.
95.         buildTree(a, segtree, 0, n - 1, 0);
```

Source Code Lazy Propagation - RQM

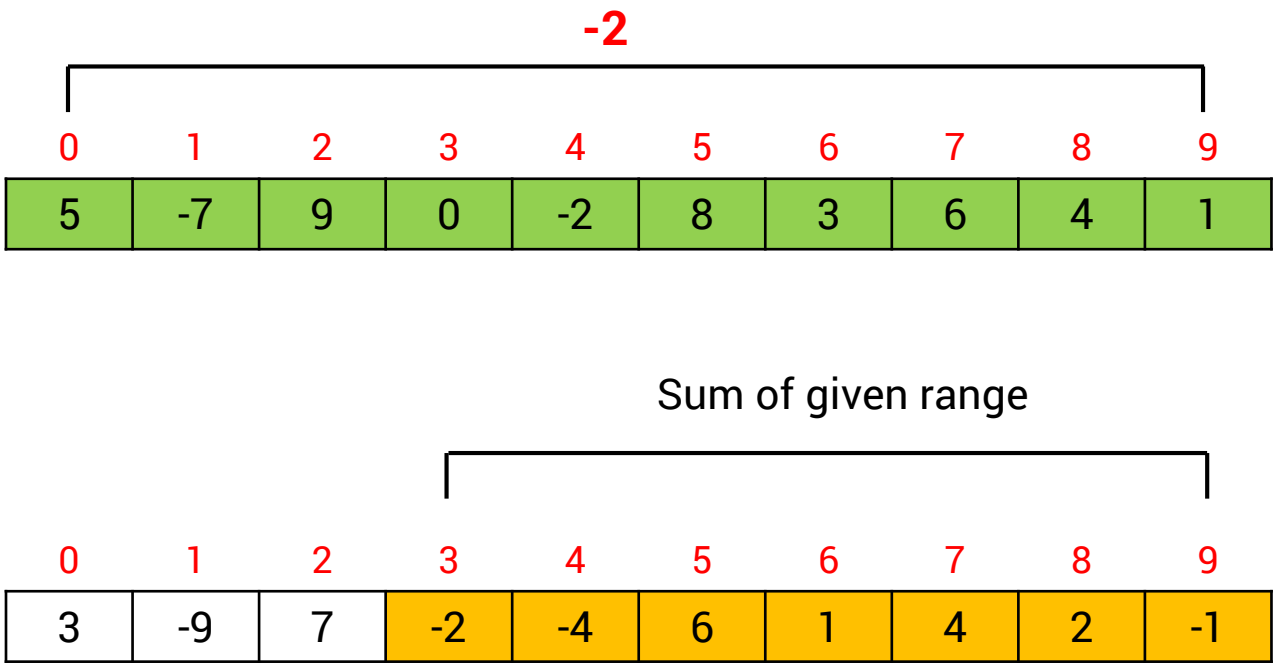
```
96.             int fromindex = 3;
97.             int toindex = 8;
98.
99.             //increment [3, 8] by 2.
100.            int delta = 2;
101.
102.            updateQuery_minRangeLazy(segtree, lazy, 0, n - 1, fromindex, toindex, delta, 0);
103.
104.            fromindex = 1;
105.            toindex = 6;
106.
107.            int res = minRangeLazy(segtree, lazy, 0, n - 1, fromindex, toindex, 0);
108.
109.            System.out.println(res);
110.
111.        }
112.    }
```

3. LAZY PROPAGATION

SUM OF GIVEN RANGE

Lazy Propagation - Sum of given range

Lazy Propagation - SGR: Cập nhật đoạn bất kỳ delta đơn vị, sau đó tính tổng của một đoạn bất kỳ trên dãy số với kỹ thuật Lazy Propagation.



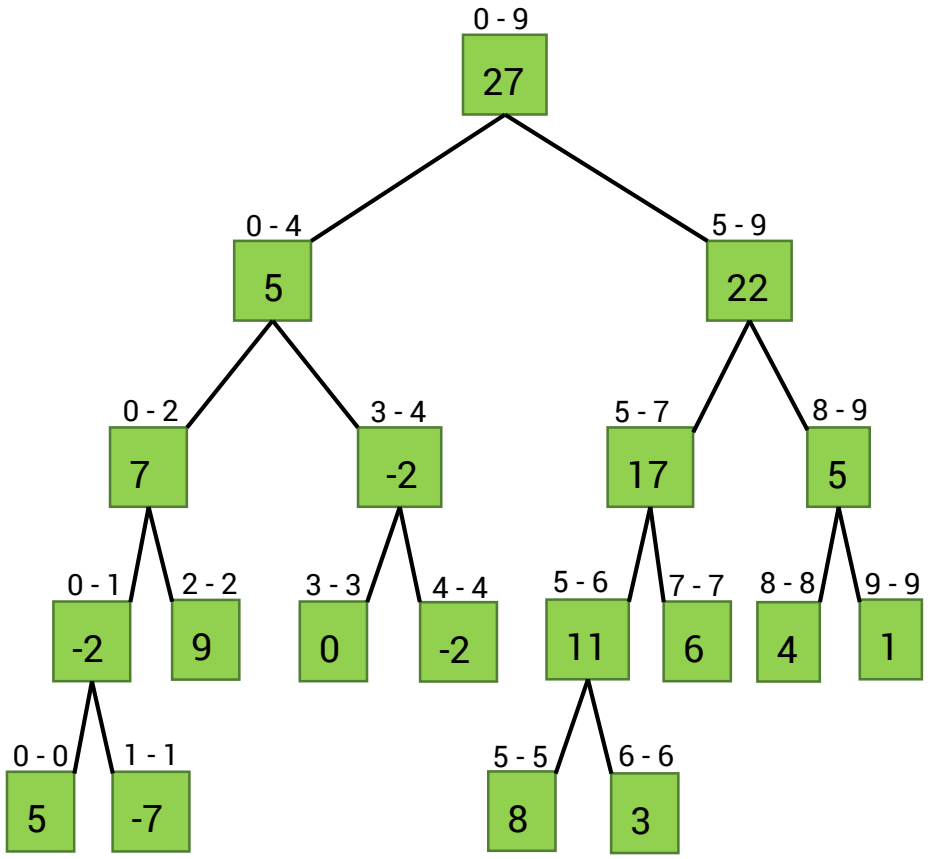
Tổng giá trị trong đoạn [3, 9]: 6

Time complexity: **O(LogN)**

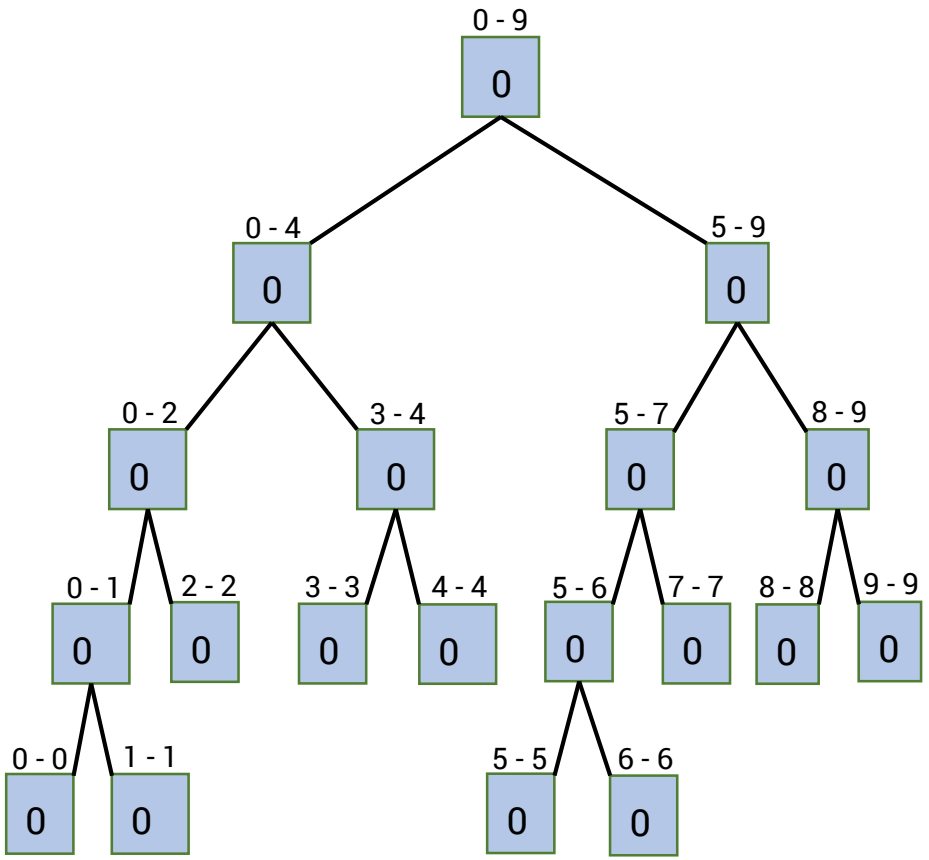
Bước 0: Xây dựng Segment Tree - SGR

0	1	2	3	4	5	6	7	8	9
5	-7	9	0	-2	8	3	6	4	1

Segment Tree

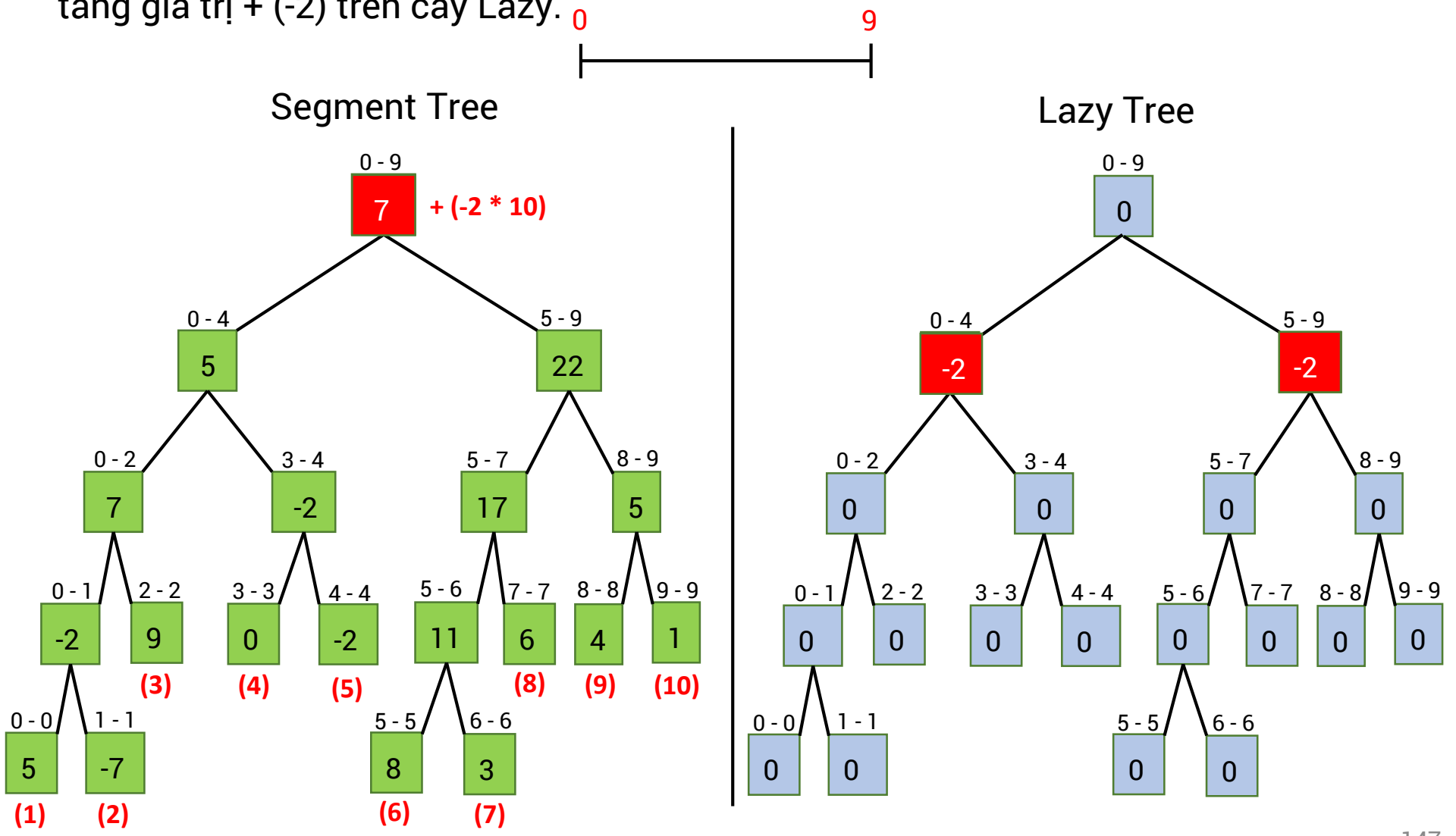


Lazy Tree



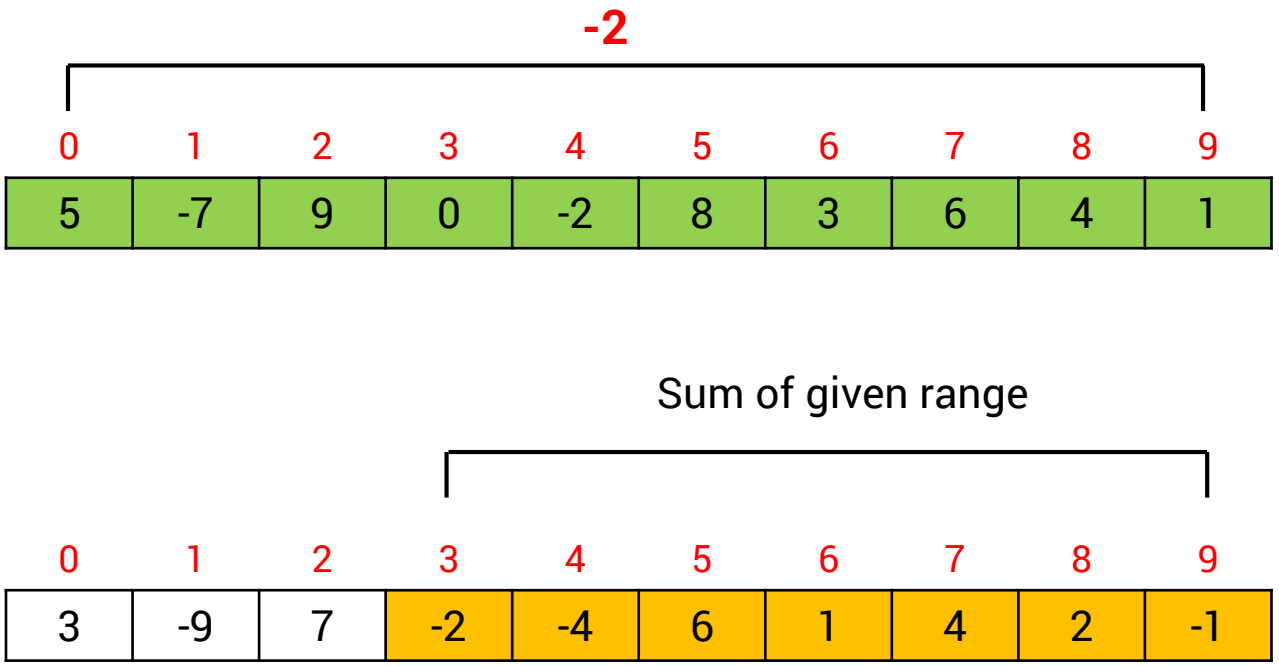
Bước 1: Cập nhật node (0, 9)

Tìm đến node (0, 9) của Segment để tăng giá trị lên + (-2 * 10). Sau đó tìm tăng giá trị + (-2) trên cây Lazy.



Lazy Propagation - Sum of given range

Lazy Propagation - SGR: Cập nhật đoạn bất kỳ delta đơn vị, sau đó tính tổng của một đoạn bất kỳ trên dãy số với kỹ thuật Lazy Propagation.



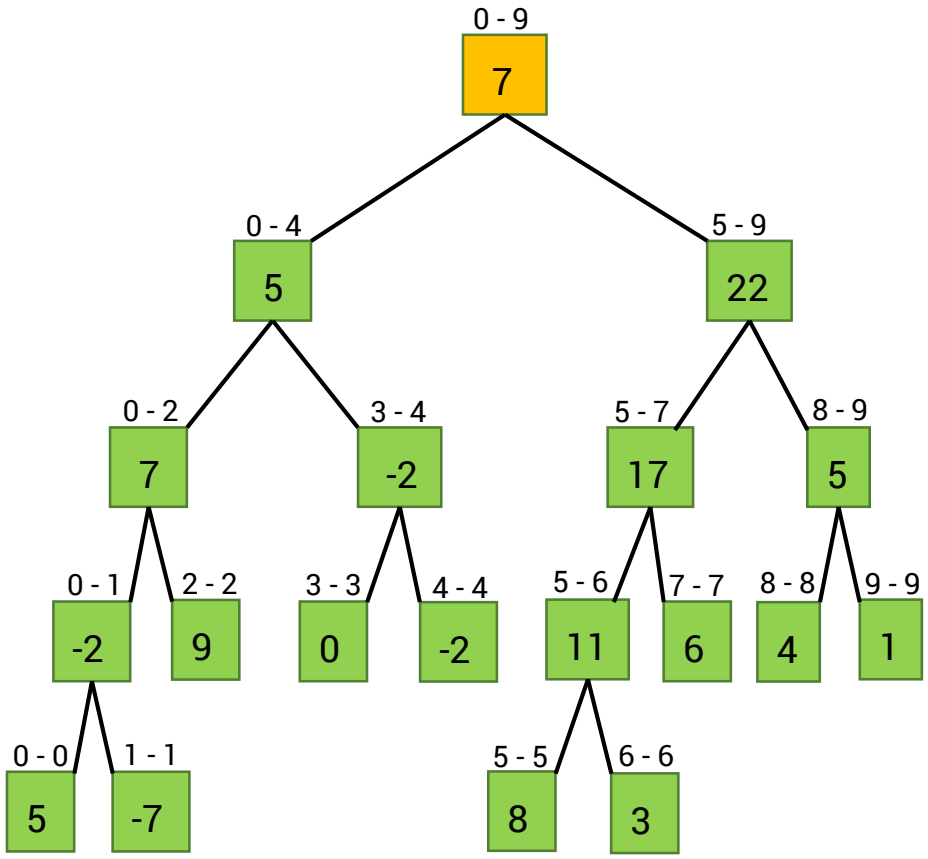
Tổng giá trị trong đoạn [3, 9]: 6

Time complexity: **O(LogN)**

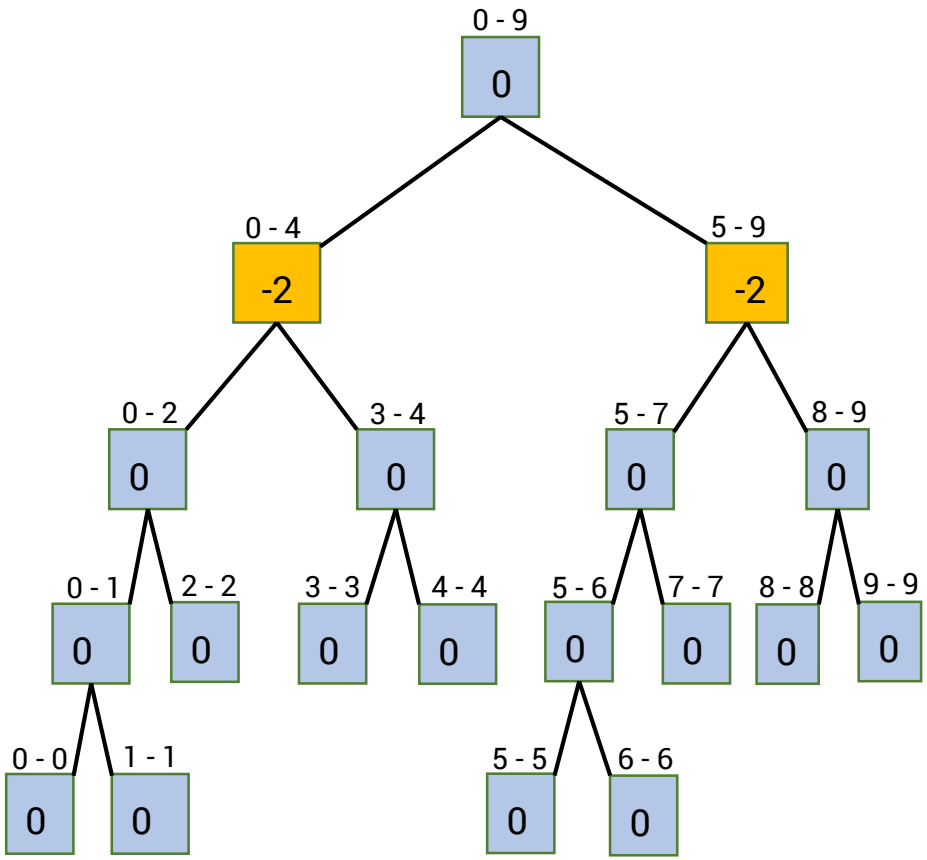
Bước 0: Cây ban đầu

0	1	2	3	4	5	6	7	8	9
3	-9	7	-2	-4	6	1	4	2	-1

Segment Tree

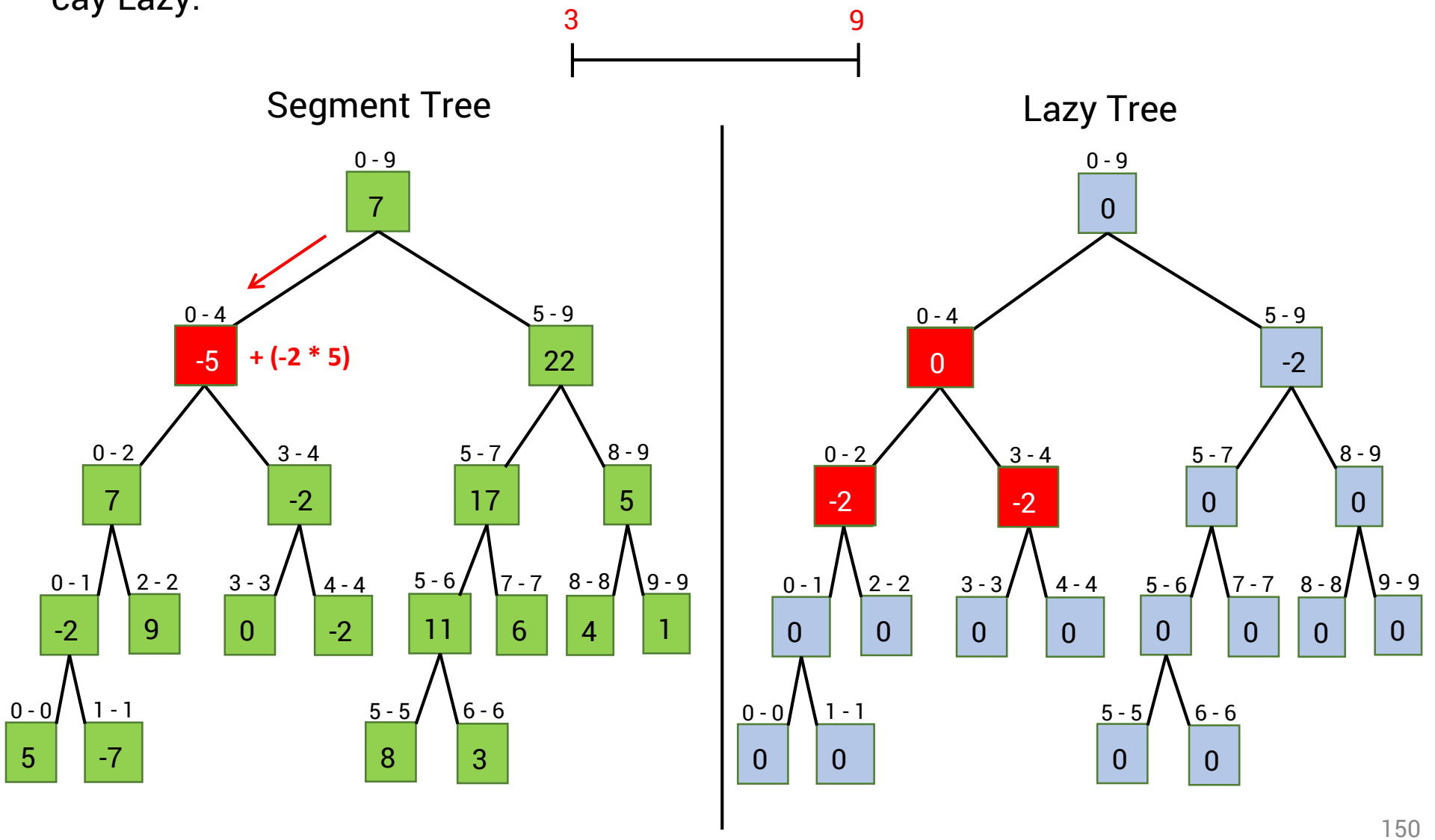


Lazy Tree



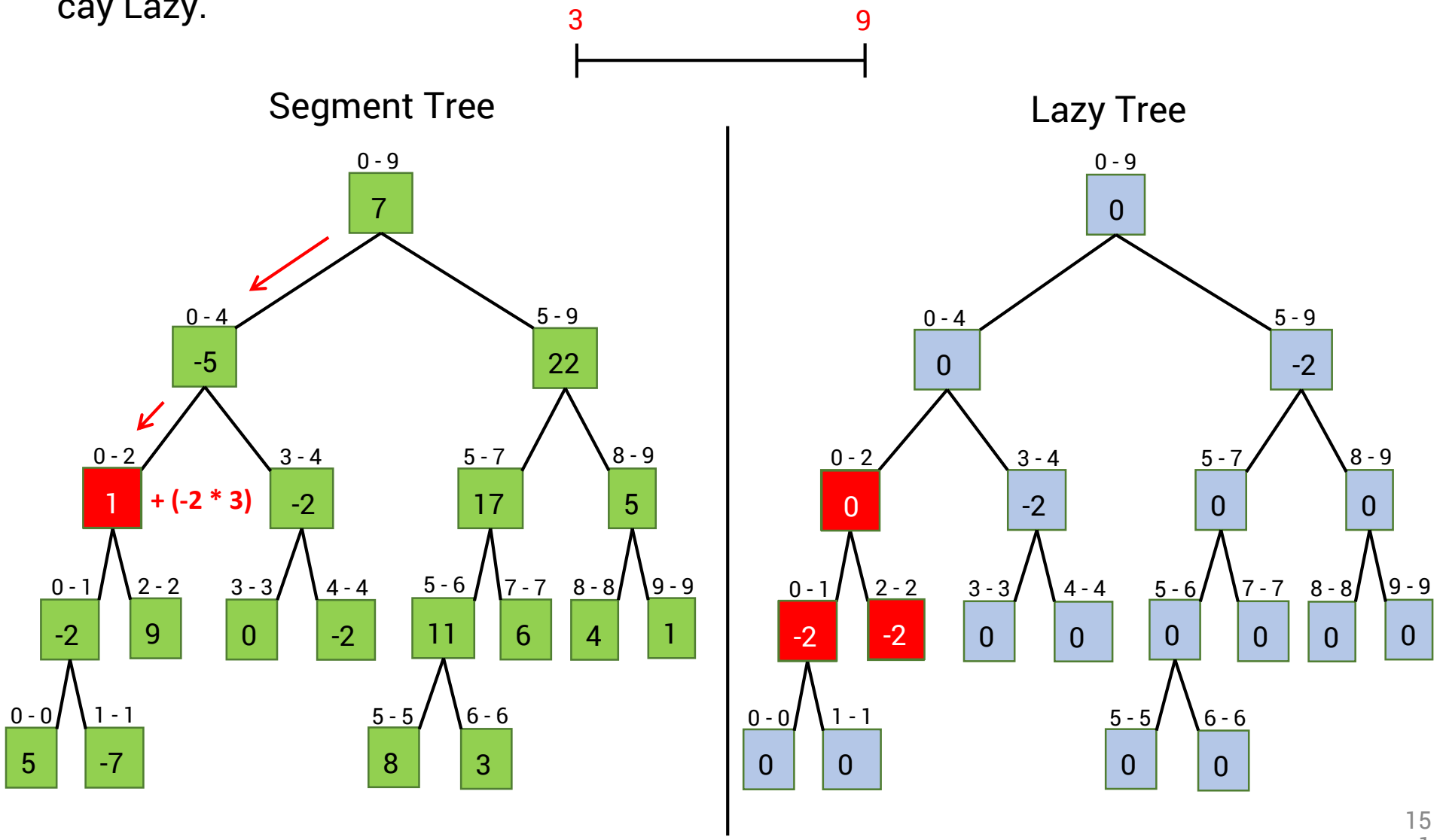
Bước 1: Tìm giá trị nhánh 1

Tìm giá trị nhánh 1, đi ngang node [0, 4] cập nhật node này và sau đó cập nhật cây Lazy.



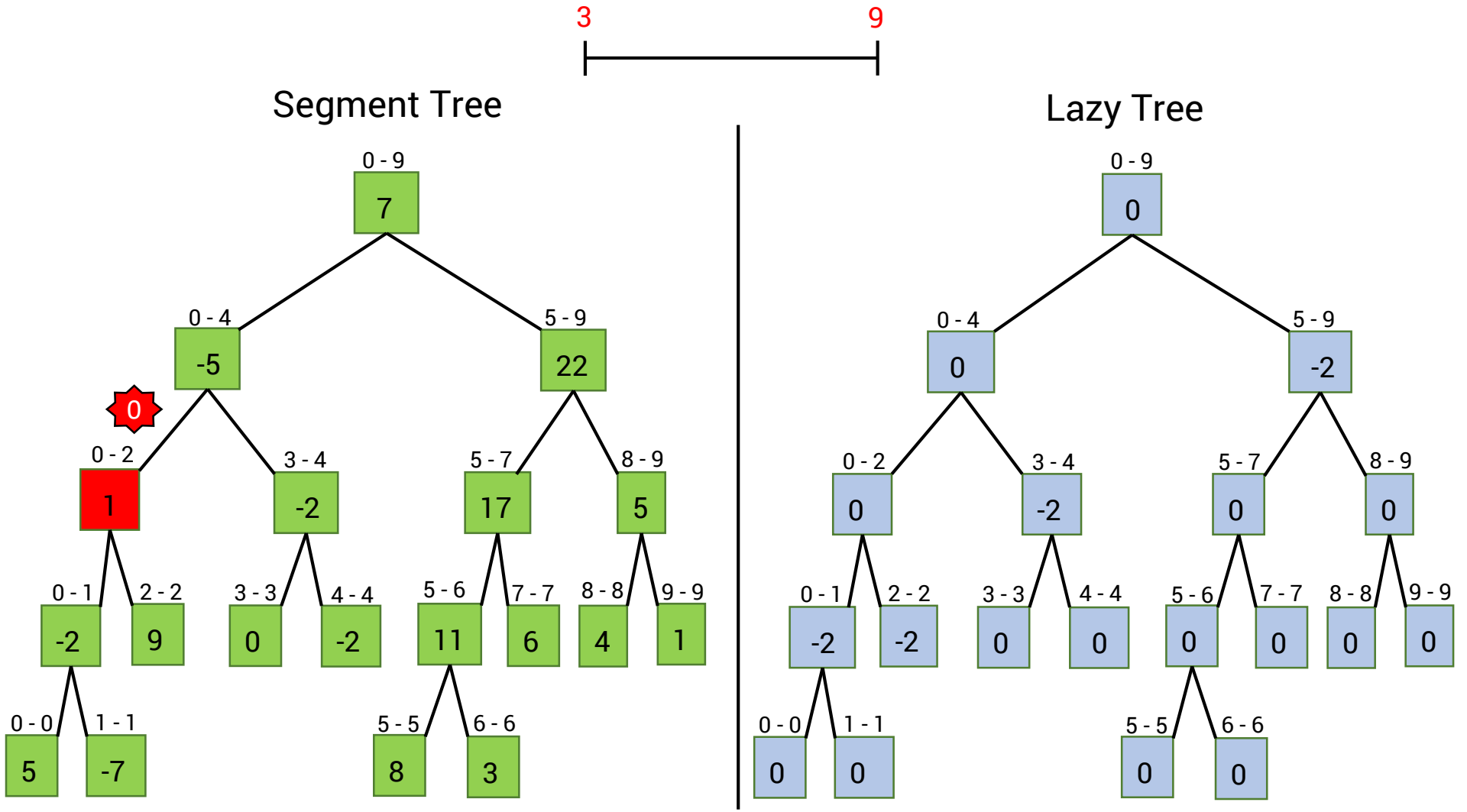
Bước 2: Tìm giá trị nhánh 1

Tìm giá trị nhánh 1, đi ngang node [0, 2] cập nhật node này và sau đó cập nhật cây Lazy.



Bước 3: Tìm giá trị nhánh 1

Giá trị nhánh 1 = 0.

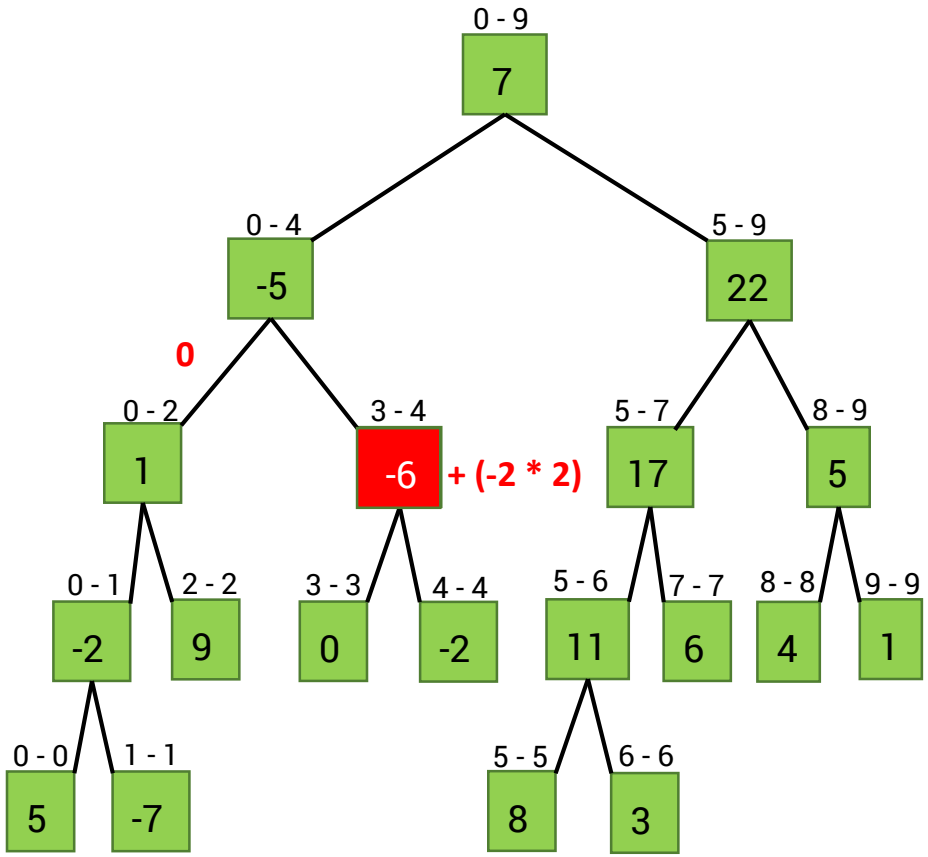


Giá trị nhánh 1: 0

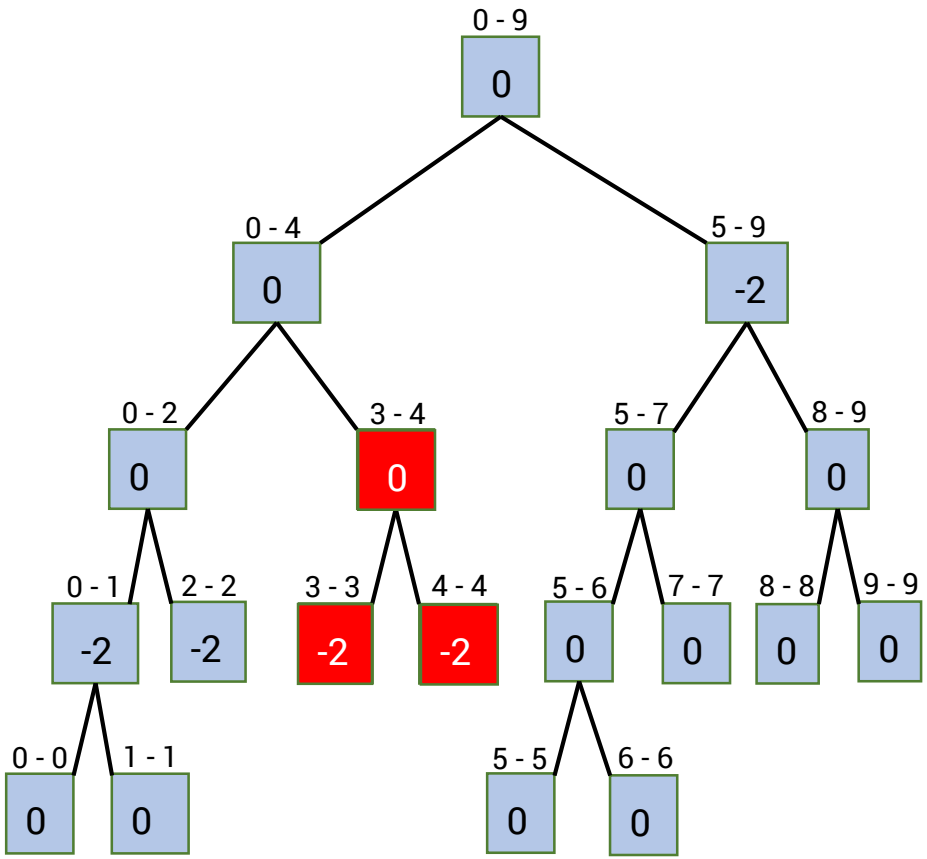
Bước 4: Tìm giá trị nhánh 2

Tìm giá trị nhánh 2, đi ngang node [3, 4] cập nhật node này và sau đó cập nhật cây Lazy.

Segment Tree



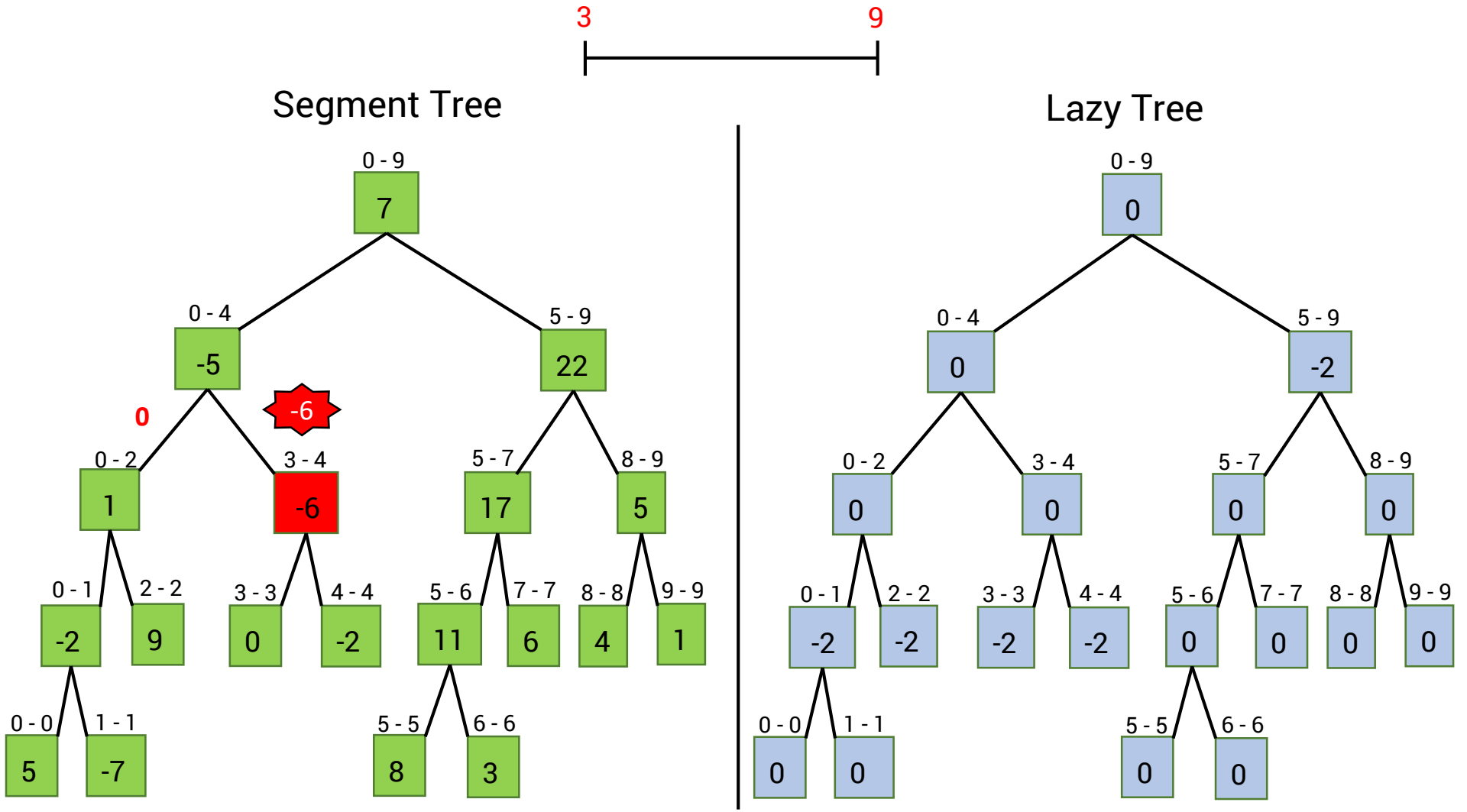
Lazy Tree



Giá trị nhánh 1: 0

Bước 5: Tìm giá trị nhánh 2

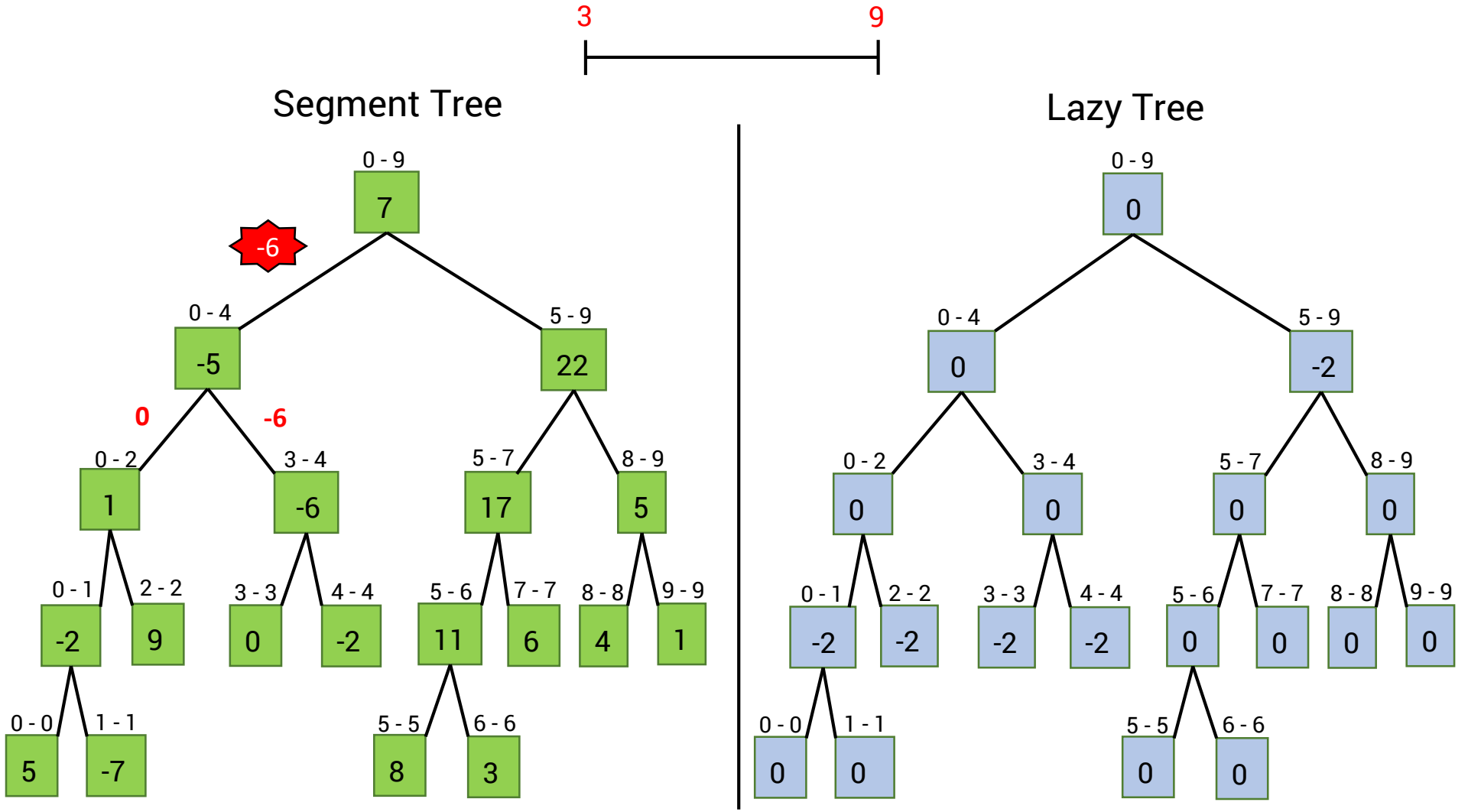
Giá trị nhánh 1 = 0.



Giá trị nhánh 2: -6

Bước 6: Giá trị toàn bộ nhánh trái

Giá trị nhánh trái = -6.

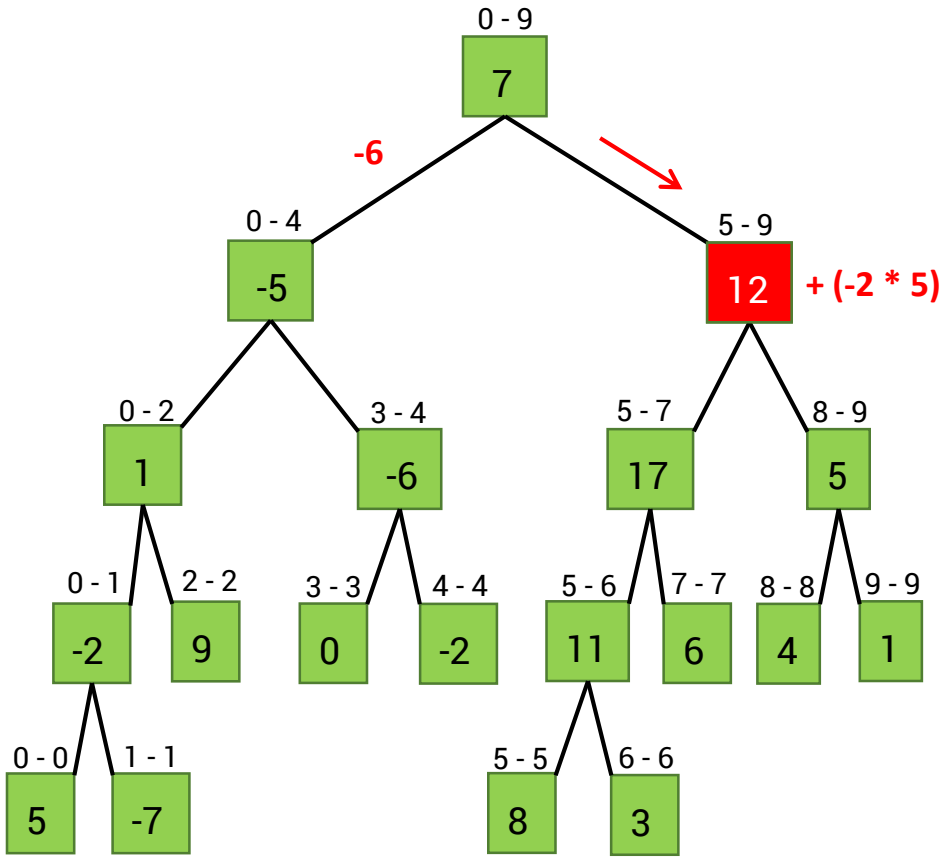


Giá trị nhánh trái: -6

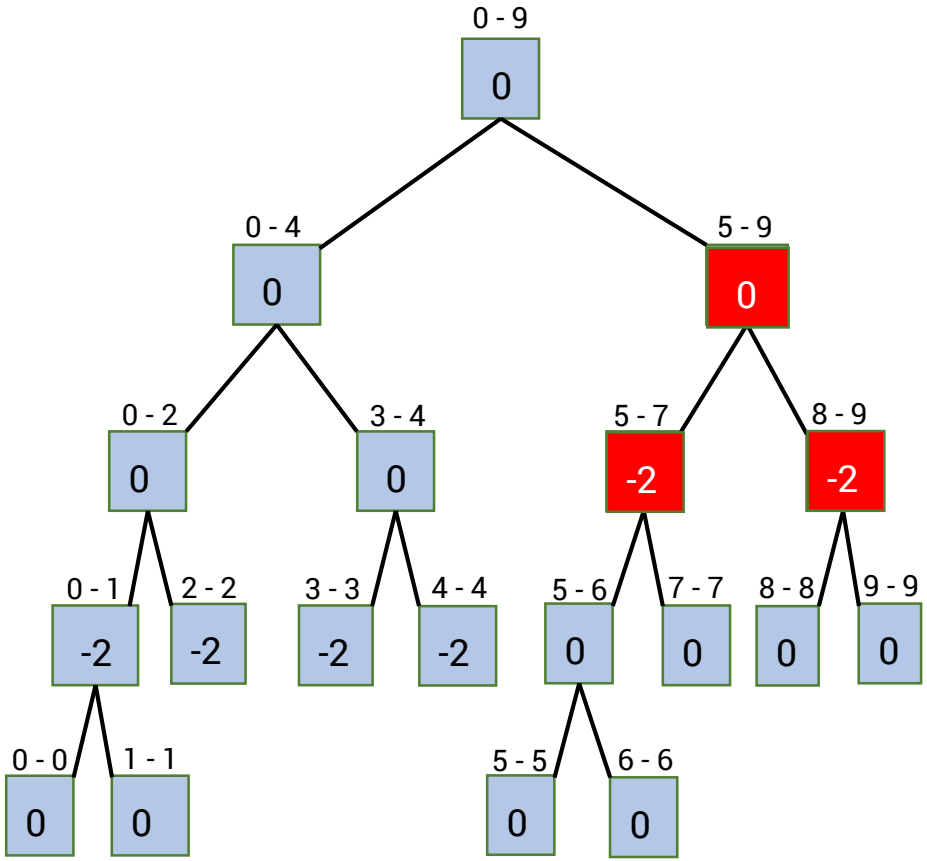
Bước 7: Giá trị nhánh 3

Tìm giá trị nhánh 3, đi ngang node [5, 9] cập nhật node này và sau đó cập nhật cây Lazy.

Segment Tree

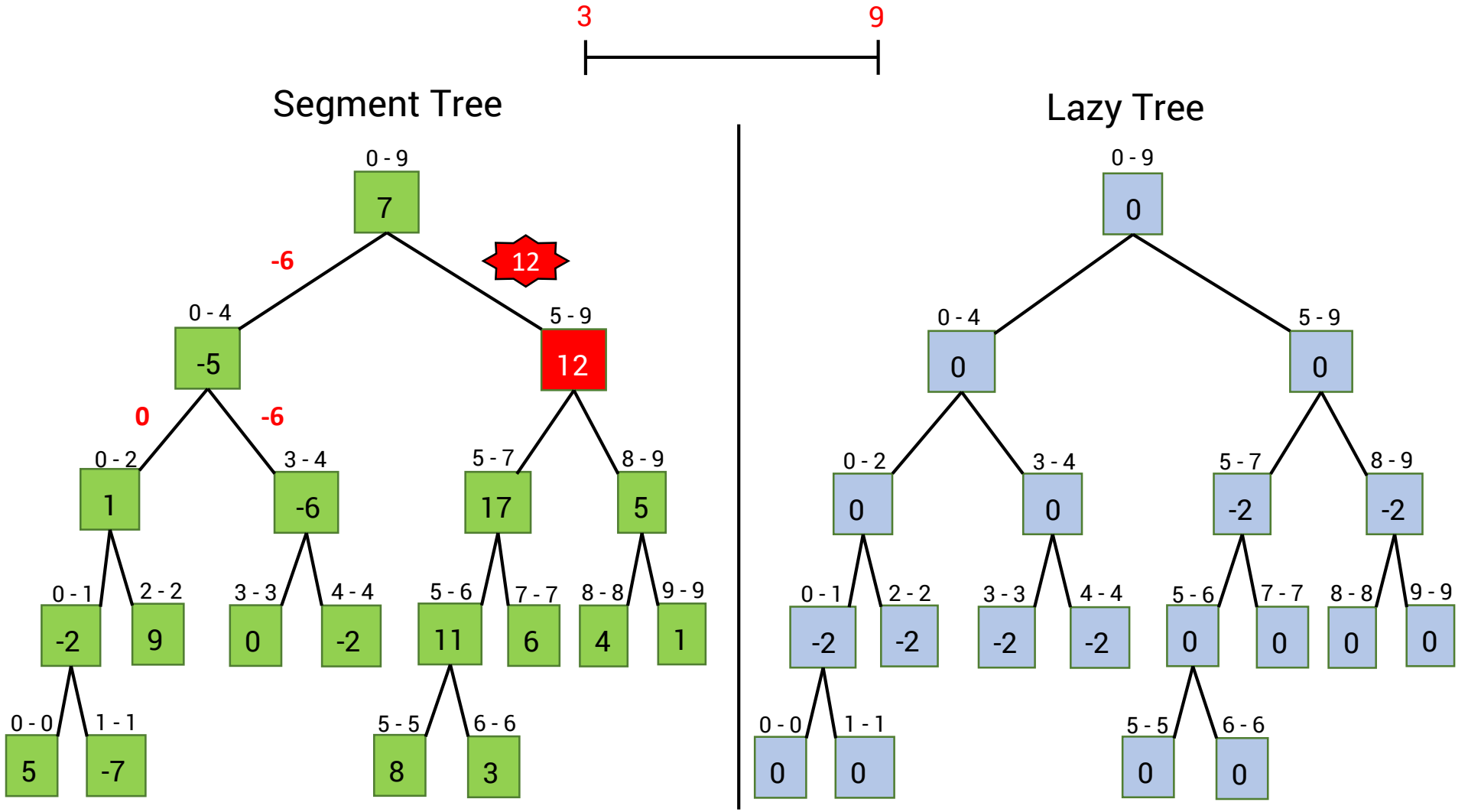


Lazy Tree



Bước 8: Giá trị nhánh 3

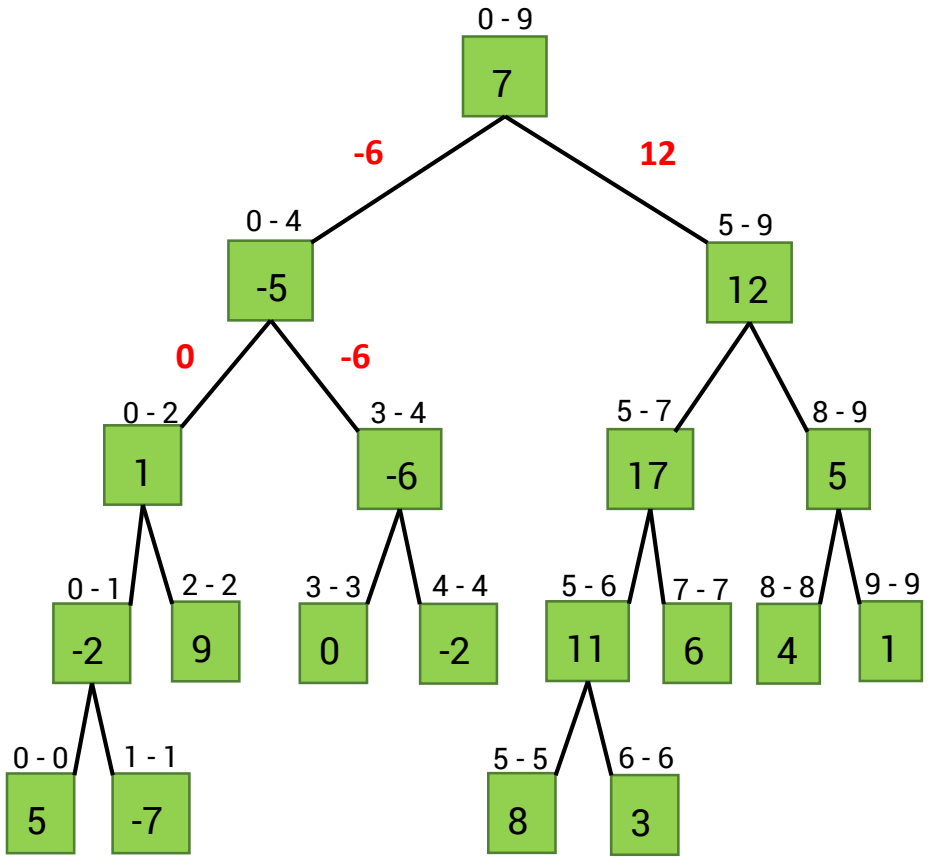
Giá trị nhánh 3 = 12.



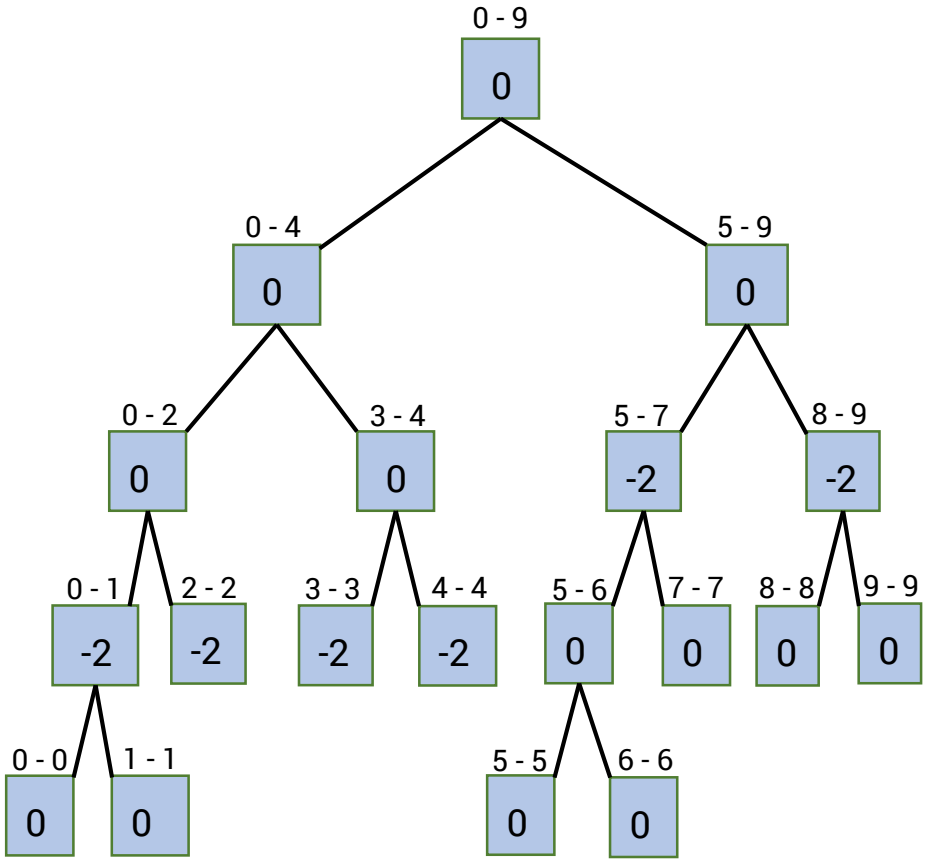
Giá trị nhánh 3: 12

Bước 9: Tổng giá trị đoạn [3, 9]

Segment Tree



Lazy Tree



Source Code Lazy Propagation - SGR

```
1.  void updateQuery_sumQueryLazy(vector<int> &segtree, vector<int> &lazy, int left, int right,
    int from, int to, int delta, int index)  {
2.      if (left > right) {
3.          return;
4.      }
5.      if (lazy[index] != 0)  {
6.          segtree[index] += lazy[index] * (right - left + 1);
7.          if (left != right)   { //not a leaf node
8.              lazy[2 * index + 1] += lazy[index];
9.              lazy[2 * index + 2] += lazy[index];
10.         }
11.         lazy[index] = 0;
12.     }
13.     //no overlap condition
14.     if (from > right || to < left)  {
15.         return;
16.     }
```

Source Code Lazy Propagation - SGR


```
17.      //total overlap condition
18.      if (from <= left && to >= right) {
19.          segtree[index] += delta * (right - left + 1);
20.          if (left != right) {
21.              lazy[2 * index + 1] += delta;
22.              lazy[2 * index + 2] += delta;
23.          }
24.          return;
25.      }
26.
27.      //otherwise partial overlap so look both left and right.
28.      int mid = (left + right) / 2;
29.      updateQuery_sumQueryLazy(segtree, lazy, left, mid, from, to, delta, 2 * index + 1);
30.      updateQuery_sumQueryLazy(segtree, lazy, mid + 1, right, from, to, delta, 2 * index + 2);
31.
32.      segtree[index] = segtree[2 * index + 1] + segtree[2 * index + 2];
33.  }
```

Source Code Lazy Propagation - SGR

```
34. int sumQueryLazy(vector<int> &segtree, vector<int> &lazy, int left, int right, int from, int to,
     int index)  {
35.     if (left > right) {
36.         return 0;
37.     }
38.     if (lazy[index] != 0)  {
39.         segtree[index] += lazy[index] * (right - left + 1);
40.         if (left != right) { //not a leaf node
41.             lazy[2 * index + 1] += lazy[index];
42.             lazy[2 * index + 2] += lazy[index];
43.         }
44.         lazy[index] = 0;
45.     }
46.
47.     //no overlap
48.     if (from > right || to < left) {
49.         return 0;
50.     }
```

Source Code Lazy Propagation - SGR

```
51.       //total overlap
52.       if (from <= left && to >= right) {
53.           return segtree[index];
54.       }
55.
56.       //partial overlap
57.       int mid = (left + right) / 2;
58.       return sumQueryLazy(segtree, lazy, left, mid, from, to, 2 * index + 1) +
59.           sumQueryLazy(segtree, lazy, mid + 1, right, from, to, 2 * index + 2);
60.   }
```

Source Code Lazy Propagation - SGR

```
1.  def updateQuery_sumQueryLazy(segtree, lazy, left, right, fr, to, delta, index):
2.      if left > right:
3.          return
4.
5.       if lazy[index] != 0:
6.          segtree[index] += lazy[index] * (right - left + 1)
7.          if left != right:
8.              lazy[2 * index + 1] += lazy[index]
9.              lazy[2 * index + 2] += lazy[index]
10.         lazy[index] = 0
11.
12.     # no overlap condition
13.     if fr > right or to < left:
14.         return
```

Source Code Lazy Propagation - SGR

```
15.     # total overlap condition
16.     if fr <= left and right <= to:
17.         segtree[index] += delta * (right - left + 1)
18.         if left != right:
19.             lazy[2 * index + 1] += delta
20.             lazy[2 * index + 2] += delta
21.         return
22.
23.     # otherwise partial overlap so look both left and right
24.     mid = (left + right) // 2
25.     updateQuery_sumQueryLazy(segtree, lazy, left, mid, fr, to, delta, 2 * index + 1)
26.     updateQuery_sumQueryLazy(segtree, lazy, mid + 1, right, fr, to, delta, 2 * index + 2)
27.     segtree[index] = segtree[2 * index + 1] + segtree[2 * index + 2]
```

Source Code Lazy Propagation - SGR

```
28.  def sumQueryLazy(segtree, lazy, left, right, fr, to, index):
29.      if left > right:
30.          return INF
31.      if lazy[index] != 0:
32.          segtree[index] += lazy[index] * (right - left + 1)
33.          if left != right:  # not a leaf node
34.              lazy[2 * index + 1] += lazy[index]
35.              lazy[2 * index + 2] += lazy[index]
36.          lazy[index] = 0
37.
38.      # no overlap
39.      if fr > right or to < left:
40.          return 0
```

Source Code Lazy Propagation - SGR

```
41.         # total overlap
42.         if fr <= left and to >= right:
43.             return segtree[index]
44.
45.         # partial overlap
46.         mid = (left + right) // 2
47.         return sumQueryLazy(segtree, lazy, mid + 1, right, fr, to, 2 * index + 2) + sumQueryLazy
    (segtree, lazy, left, mid, fr, to, 2 * index + 1)
```

Source Code Lazy Propagation - SGR

```java
private static void updateQuery_sumQueryLazy(int[] segtree, int[] lazy, int left, int right,
 int from, int to, int delta, int index) {
        if (left > right) {
            return;
        }

        if (lazy[index] != 0) {
            segtree[index] += lazy[index] * (right - left + 1);
            if (left != right) { //not a leaf node
                lazy[2 * index + 1] += lazy[index];
                lazy[2 * index + 2] += lazy[index];
            }
            lazy[index] = 0;
        }

        //no overlap condition
        if (from > right || to < left) {
            return;
        }
```

Source Code Lazy Propagation - SGR


```
19.         //total overlap condition
20.         if (from <= left && to >= right) {
21.             segtree[index] += delta * (right - left + 1);
22.             if (left != right) {
23.                 lazy[2 * index + 1] += delta;
24.                 lazy[2 * index + 2] += delta;
25.             }
26.             return;
27.         }
28.
29.         //otherwise partial overlap so look both left and right.
30.         int mid = (left + right) / 2;
31.         updateQuery_sumQueryLazy(segtree, lazy, left, mid, from, to, delta, 2 * index + 1);
32.         updateQuery_sumQueryLazy(segtree, lazy, mid + 1, right, from, to, delta, 2 * index + 2);
33.
34.         segtree[index] = segtree[2 * index + 1] + segtree[2 * index + 2];
35.     }
```

Source Code Lazy Propagation - SGR

```
36.     private static int sumQueryLazy(int[] segtree, int[] lazy, int left, int right, int from, int
    to, int index) {
37.         if (left > right) {
38.             return 0;
39.         }
40.         if (lazy[index] != 0) {
41.             segtree[index] += lazy[index] * (right - left + 1);
42.             if (left != right) { //not a leaf node
43.                 lazy[2 * index + 1] += lazy[index];
44.                 lazy[2 * index + 2] += lazy[index];
45.             }
46.             lazy[index] = 0;
47.         }
48.
49.         //no overlap
50.         if (from > right || to < left) {
51.             return 0;
52.         }
```

Source Code Lazy Propagation - SGR

```
53.         //total overlap
54.         if (from <= left && to >= right) {
55.             return segtree[index];
56.         }
57.
58.         //partial overlap
59.         int mid = (left + right) / 2;
60.         return sumQueryLazy(segtree, lazy, left, mid, from, to, 2 * index + 1) +
61.                 sumQueryLazy(segtree, lazy, mid + 1, right, from, to, 2 * index + 2);
62.     }
```

Hỏi đáp